Eötvös Loránd Tudományegyetem
Természettudományi Kar

Az adatbányászat hibrid modelljei

Szakdolgozat

Olexó Tünde
Matematika BSc,
matematikai elemző szakirány

Témavezető: Lukács András
Számítógéptudományi Tanszék

2018
Budapest

Tartalomjegyzék

1. Előszó **5**

1.1. Bevezetés 5

1.2. A hibrid modellek története 6

1.3. A hibrid modellek általános bemutatása 8

2. Boosting **11**

2.1. Boosting általános algoritmusa és alapelve 11

2.2. AdaBoost 13

2.3. MadaBoost változat a túlsúlyozás mérséklésére 16

2.4. Gradient Boosting és XGBoost 18

2.5. Veszteség- és kockázatfüggvények 21

2.6. Kitekintés további Boosting algoritmusokra 25

3. Bagging **29**

4. Kombinációs módszerek **31**

4.1. Kombinációs módszerek általános bemutatása 31

4.2. Átlag 33

4.2.1. Egyszerű átlag 33

4.2.2. Súlyozott átlag 33

4.3. Szavazás 35

4.3.1. Abszolút többségi szavazás 36

4.3.2. Relatív többségi szavazás 36

4.3.3. Súlyozott szavazás 37

4.3.4. Lágy szavazás 37

4.4. Tanuló általi kombinálás 39

A. Szemléltető modellek, kódok **41**

A.1. Táblás modell Boosting algoritmusokhoz 41

A.2. AdaBoost algoritmus 44

A.3. Táblás modell az AdaBoost algoritmussal 45

A.4. 10 kicsi indián feladat a GBM szemléltetéséhez 49

A.5. A 10 kicsi indián feladat grafikus megoldása 50

A.6. 10 kicsi indián a GBM algoritmussal 54

A.7. Az AdaBoost, MadaBoost modell programjának forráskódja 59

B. Jelölések, jegyzékek **66**

B.1. Jelölések 66

B.2. Magyar –angol szójegyzék 68

B.3. Ábrák jegyzéke 70

B.4. Algoritmusok jegyzéke 71

B.5. Irodalomjegyzék 72

Köszönetnyilvánítás

Hálásan köszönöm témavezetőmnek, Lukács Andrásnak a számtalan konzultációs alkalmat, segítőkészségét, rugalmasságát, ötleteit, észrevételeit a dolgozat érthetőbbé és áttekinthetőbbé tételéhez. Köszönöm a Valószínűségelméleti és Statisztika Tanszéknek, különösen Márkus Lászlónak, Németh Lászlónak, Varga Lászlónak és Kornyik Miklósnak, hogy az adatbányászathoz szükséges egy- és többdimenziós statisztikai és valószínűségszámítási alapokat megtanították, megszerettették velem. Hálás vagyok a Számítógéptudományi Tanszéknek, elsősorban Sziklai Péternek, Fekete Istvánnak, Héger Tamásnak, Wolosz Jánosnak és Recski Andrásnak a gráfelmélet megismertetéséért. Köszönöm családtagjaimnak az évek során nyújtott támogatást.

1. fejezet

Előszó

1.1. Bevezetés

A dolgozat célja az adatbányászati hibrid modellek elméleti hátterének megismertetése, szemléltetése példákon és ábrákon keresztül. A szakdolgozat feltételezi az adatbányászat alapjainak ismeretét, így az adatbányászati alapdefiníciók, fogalmak csak említés szintjén szerepelnek benne. Az alapozó ismeretek megszerzéséhez ajánlott Tan-Steinbach-Kumar: *Adatbányászat* című könyve [23].

Hibrid modellek alatt olyan összetett adatbányászati modelleket (eljárásokat) értünk, amelyek több modellt építenek ugyanarra a feladatra, majd eredményeiket kombinálják a kombinációs módszerek – például többségi szavazás, súlyozott átlag – valamelyikével. Az épített modellek – például döntési fák, neurális hálók – lehetnek azonosak (homogén) és különbözőek (heterogén) egyaránt.

A szakdolgozat keletkezéséig nem született magyar nyelvű szakirodalom a hibrid modellek témakörében, ezért a dolgozatban használt fogalmak nagy részét saját fordításban közlöm. A felhasznált – magyarra lefordított – fogalmak angol megfelelőjükkel a függelékben, azon belül *a B.2.* részben találhatók.

Tartalmi összefoglaló:

Az **1. fejezet** a hibrid modellek rövid történetét, alkalmazási területeit, alapgondolatát, fő csoportjainak általános bemutatását tartalmazza.

A **2. fejezet** az általános Boosting algoritmust és annak különböző paraméterezésével kapott AdaBoost, Gradient Boosting, és XGBoost algorimust mutatja be. Emellett kitekintést nyújt más Boosting algoritmusokra (LogitBoost, LPBoost) is. Tartalmazza az

említett modellekben alkalmazható veszteség (L_1, L_2, Huber-, Tukey-)- és kockázatfüggvények ismertetését, összehasonlítását alkalmazhatóságuk, meghatározó tulajdonságaik – például deriválhatóság – alapján. Az AdaBoost túltanulási tulajdonságával kapcsolatos megfigyeléseket, eredményeket is ismerteti.

A **3. fejezet** a Bagging algoritmust mutatja be a hibrid modellek tanítási sorrend szerinti csoportosításának alapfogalmai – szekvenciális (például AdaBoost, MadaBoost, Gradient-Boosting) és párhuzamos hibrid modellek (például Bagging)– mellett.

A **4. fejezet** a kombinációs módszerek általános bemutatásával, majd az átlagoló kombinációs módszerekkel (egyszerű és súlyozott átlag), szavazó kombinációs módszerekkel (abszolút többségi, lokális többségi, súlyozott és lágy szavazás) foglalkozik. Emellett a témakörben alkalmazott fogalmakat (éles címke, osztályvalószínűség) definiálja, súlyozott átlag és szavazás esetén az optimális súlyt meghatározza. Kitekint további kombinációs módszerekre (tanuló általi kombinálás) is.

Az **A.Függelék** a Boosting algoritmusok (AdaBoost, Gradient Boosting) könnyebb megértéséhez tartalmaz egyszerű feladatokat ábrákkal, 10 adatpontra. Ezek Shapire mintafeladatainak továbbfejlesztései [19, 1.2.1 p. 7-10]. A feladatokhoz tartozó segédprogramok kódjai is itt találhatók.

A **B.Függelék** a dolgozat során használt jelölések összefoglalását, magyar-angol szójegyzéket, továbbá az ábrák, algoritmusok és irodalom jegyzékét tartalmazza.

1.2. A hibrid modellek története

A hibrid modellek kialakulásának pontos idejét nehéz meghatározni. Több hibrid modell kezdemény született, amiről érdemes szót ejtenünk. Sokan Tukey 1977-es cikkét tartják kiindulópontnak, aki olyan két fázisú regressziós modellt konstruált, amelyben a második egyenest az első reziduálisaihoz illesztette [22]. Mások Dasarathy modelljét tartják kezdetnek, aki 1979-ben olyan osztályozót készített, ahol a tulajdonságokat *oszd meg és uralkodj* elv alapján több kisebb csoportba osztotta, és ezeket egyesítette [3]. Mindenesetre mivel a hibrid módszerek elmélete a nehezen tanulható feladatok és könnyen tanulható feladatok azonosságának kérdésén alapszik (Kearns és Valiant, 1984), ezért legáltalánosabban elfogadott vélemény, hogy a hibrid módszerek kialakulásának időpontja 1984. Ekkor tette fel ugyanis Valiant a kérdést, hogy a nehezen (erősen) és könnyen (gyengén) tanulható problémák osztálya azonos-e [24].

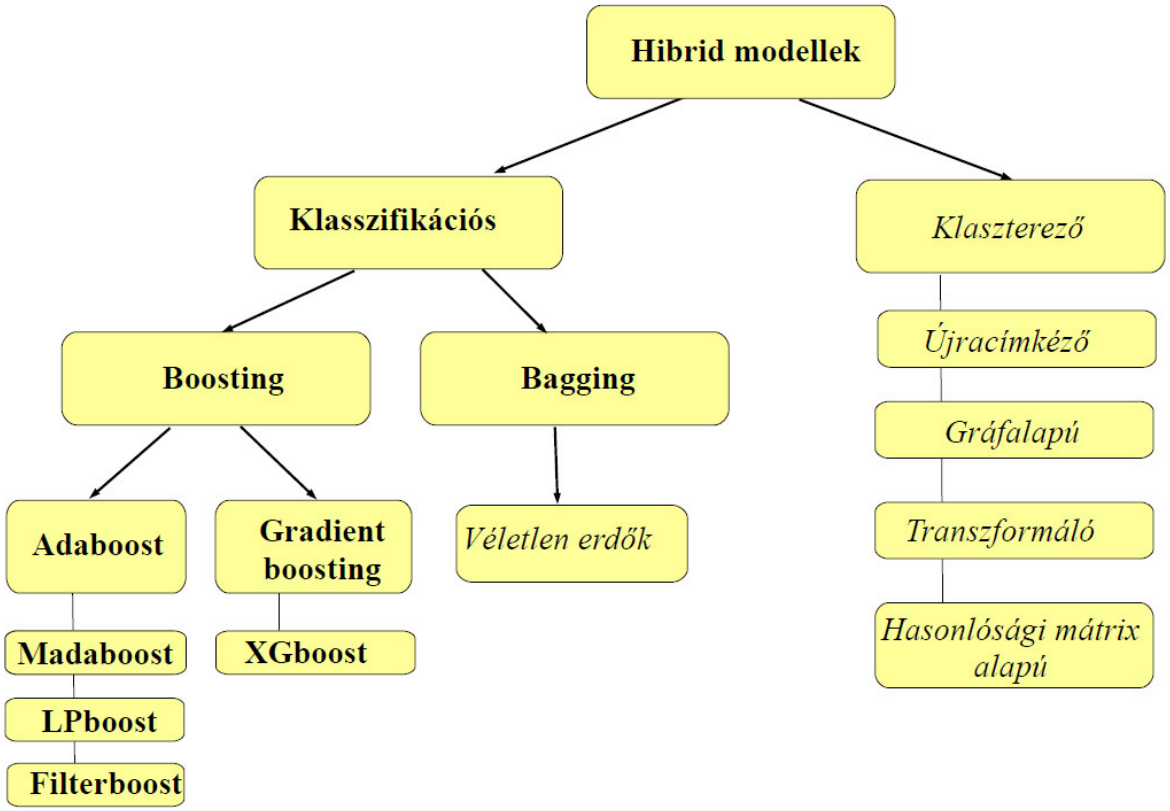
Egy probléma nehezen tanulható, ha az adathalmaz bármely eloszlására $\forall\ \varepsilon > 0$,

$\delta > 0$ -hoz $P > (1 - \delta)$ valószínűséggel található egy $err \leq \varepsilon$ hibával közelítő hipotézis. A probléma könnyen tanulható, ha a $\forall\, \varepsilon > 0$ feltételt $\forall\, \varepsilon > \frac{1}{2} - \gamma$ $(\gamma \geq 0)$ feltételre cseréljük [19]. (Feladat, más néven probléma alatt azt értjük, hogy egy $\mathcal{D}$ eloszlású független minta elemei alapján szeretnénk becsülni a célváltozó értékét.) Megjegyezzük, hogy a gyakorlatban a minta függetlensége ritkán fordul elő. A gyengén és erősen tanulható feladatok halmazának azonosságát 1990-ben Schapire igazolta a Boosting algoritmussal. (Vagyis gyenge tanulók kombinálásával egy erős tanulóhoz juthatunk.) A bizonyítás maga egy olyan konstrukció, ami gyenge tanulók kombinálásával erős tanulót állít elő. Ez a kezdeti Boosting algoritmus (modell) nem volt a gyakorlatban használható. Schapire és Freund 1995-97 körül mutatták be az AdaBoost első implementálható változatát. Ennek továbbfejlesztése az AdaBoost.M1 – diszkrét AdaBoost – algoritmus, az AdaBoost legismertebb változata *(Friedmann, 2000)*.

Az adatbányászati hibrid modellek iránti komolyabb érdeklődés kezdete a 90-es évek elejére tehető. Azóta széles körben foglalkoznak fejlesztésükkel. Így nem csak adatbányászati versenyeken *(KDD Cup, Netfix Prize, Kaggle)* kerülnek elő, hanem mindennapjaink részévé váltak. Gyakran használt szoftverek is alapulnak rajtuk, például Viola és Jones (2001, 2004) arcdetektáló szoftvere AdaBooston alapul [26]. Ezt építették be a máig használt fényképezőgépekbe. Érdemes megjegyezni, hogy Viola és Jones szoftvere csak azt ismeri fel, emberi arcról van-e szó, nem képes *konkrét arcok* azonosítására.

Orvosi diagnosztika terén említésre méltó Zhou munkássága, aki 2002-ben tüdőrák diagnosztizálásra alkalmas, tanulók általi kombináláson alapuló kétszintes modellt alkotott meg.

A hibrid modellek fajtái az alábbi ábrán láthatók. Ezek egy részét foglalja össze a dolgozat, terjedelmi határok miatt a teljesség igénye nélkül.

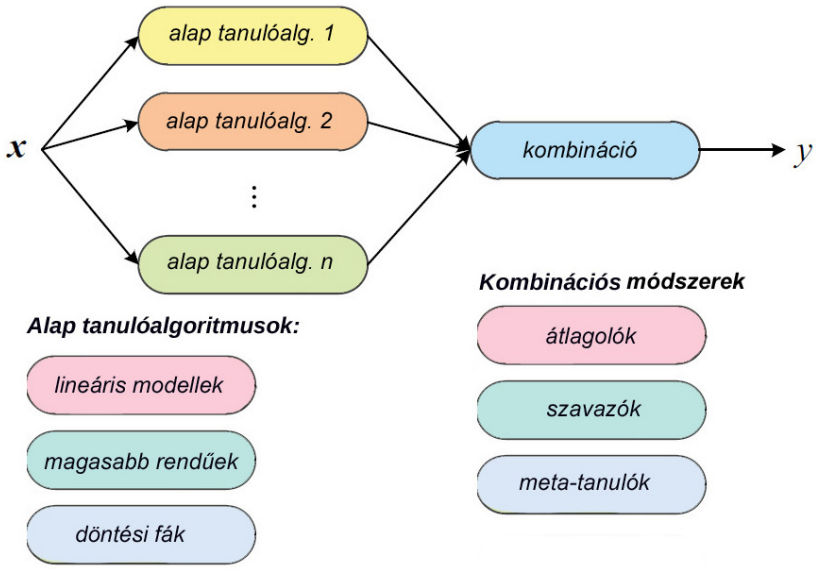


1.1. ábra. A hibrid modellek fajtái

1.3. A hibrid modellek általános bemutatása

Hibrid modellek alatt olyan összetett adatbányászati modelleket értünk, amelyek alaptanulók – más néven alapmodellek, gyenge tanulók – egy sorozatát tanítják egy azonos feladaton, majd eredményeiket, a célváltozó becsült értékeit kombinálják a kombinációs módszerek valamelyikével.

A hibrid modelleknél használt kombinációs módszer lehet például az abszolút többségi, lágy szavazás, lineáris kombináció és súlyozott átlag. A kombinálási módszereket *a 4.* fejezetben ismertjük. Az épített modellek például döntési fák, neurális hálók lehetnek azonosak (homogén) és különbözőek (heterogén) egyaránt. Gyenge – más néven alap- vagy primitív – tanulón olyan modelleket értünk, amik jobban teljesítenek mint a véletlenszerű tippelés. Az alábbi ábrán a hibrid modellek általános felépítése látható, a benne szereplő paraméterek leggyakoribb értékeivel.



1.2. ábra. A hibrid modellek felépítése

Általában azonos típusú alaptanulókból álló modellt generálunk a homogenitás elérésének érdekében, de vannak olyan módszerek is, melyek többféle modell segítségével állítják össze a gyenge tanulók halmazát, így heterogén sorozatot (halmazt) eredményeznek. A heterogén esetben – mivel nincs egy konkrét algoritmus, ami a tanulókat meghatározza –, a szakirodalomban az *alaptanuló* helyett a *független tanuló* megfogalmazást részesítik előnyben. A hibrid módszerek népszerűségének oka, hogy könnyen előállítható gyenge tanulókat – melyek lehetnek relatíve elég erősek is – erős tanulókká tudnak kombinálni. Vagyis a kombinált tanulók együtt még jobban teljesítenek, mint külön-külön, így pontosabb becslését adják a célváltozónak.

A hibrid módszerek alkalmazásának két alapvető célja lehet:

gyorsítás: minél gyorsabban szeretnénk megoldáshoz jutni az adathalmaz nagy elemszáma, vagy dinamikus változása miatt. Például tőzsde modellezése esetén az adathalmaz elemei –bemeneti magyarázóváltozók– folyamatosan frissülnek, így elengedhetetlen a gyors eredmény annak használhatósága érdekében.

pontosítás: adott erőforrást szeretnénk felhasználni minél pontosabb modell építésére. Kevésbé érzékeny a zajra, gyakran a torzítást, nagy szórás hatását kiküszöböli, mivel egy modell hibájának a hatása a hibrid modell eredményére kicsi, feltéve, hogy sok modellünk van.

Általában a pontosság növelése és gyorsítás egymással ellentétesen hat egy hibrid modell esetén. (Kivétel az XGBoost, de itt főként *a 2.4.* fejezetben ismertetett informatikai újítások miatt jutunk gyorsabb és pontosabb modellhez.)

2. fejezet

Boosting

2.1. Boosting általános algoritmusa és alapelve

A Boosting algoritmusok olyan algoritmuscsaládba tartoznak, melyek képesek gyenge tanulókat erős tanulókká alakítani. (Az angol *boost* gyorsítást jelent.)

A Boosting algoritmust általános paraméterekkel *a 2.1.* algoritmus mutatja be. A paraméterek különböző megválasztásával más-más algoritmushoz jutunk. Például ha abszolút többségi szavazást és exponenciális hibafüggvényt alkalmazunk, az AdaBoost algoritmushoz jutunk.

Általános Boosting algoritmus

Bemenet:

A tanításhoz használt mintavételezés eloszlása: $\mathcal{D}$;

Alap tanuló algoritmus: $\mathcal{L}$;

Tanulási folyamat ciklusszáma: T.

Folyamat:

1. $\mathcal{D}_1 = \mathcal{D}$. % Eloszlás inicializálása
2. **for** $t = 1, \ldots, T$:
3. $h_t = \mathcal{L}(\mathcal{D}_t)$; % $\mathcal{D}_t$ eloszlással gyenge tanuló tanítása
4. $\varepsilon_t = P_{x \sim D_t}(h_t(x) \neq f(x))$; % h_t hibájának kiértékelése
5. $\mathcal{D}_{t+1} = Adjust_Distribution(\mathcal{D}_t, \varepsilon_t)$ % Eloszlás újraszámolása
6. **end**

Kimenet:

$H(x) = Combine_Outputs(\{h_1(x), \ldots, h_t(x)\})$ % A $(h_1, \ldots, h_t)$ tanulók kombinálása.

2.1. algoritmus. Általános Boosting algoritmus

2.2. AdaBoost

A 2.1. fejezetben megadott általános Boosting algoritmus paramétereinek alábbi beállításával az AdaBoost.M1 – ez maradt meg AdaBoostként a köztudatban – algoritmushoz jutunk *(Friedmann, 2000)*.

Paraméterek definiálása:

- adathalmaz típusa: bináris, $x \in \{-1, +1\}$
- cél: exponenciális veszteség minimalizálása (Adjust_Distribution)
- kombinációs módszer: alaptanulók additív súlyozott kombinációja (Combine_Outputs)

Az exponenciális veszteség használatát a klasszifikációs hiba minimalizálásának a maximum-likelihood becsléssel való kapcsolata és konzisztenciája indokolja.

Megmutatható, hogy az exponenciális veszteség minimalizálása esetén a klasszifikációs hibát is minimalizáljuk, tehát a differenciálható exponenciális veszteség megfeleltethető a nem differenciálható klasszifikációs hiba helyettesítésével modellünk esetén.

AdaBoost.M1 algoritmus

Bemenet:

Adathalmaz: $D = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$;
Alap tanuló algoritmus: $\mathcal{L}$;
Tanulási folyamat ciklusszáma: T.

Folyamat:

1. $\mathcal{D}_1(x) = \frac{1}{m}$. % Súlyok eloszlásának inicializálása
2. **for** $t = 1, \ldots, T$:
3. $h_t = \mathcal{L}(D, \mathcal{D}_t)$; % h_t klasszifikátor tanítása a $\mathcal{D}_t$ eloszlású D alaphalmazon
4. $\varepsilon_t = P_{x \sim D_t}(h_t(x) \neq f(x))$; % h_t hibájának kiértékelése
5. **if** $\varepsilon_t > 0.5$ **then break**
6. $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\varepsilon_t}{\varepsilon_t} \right)$; % h_t tanuló súlyának meghatározása
7. $\mathcal{D}_{t+1}(x) = \frac{\mathcal{D}_t(x)}{Z_t} \times \begin{cases} \exp(-\alpha_t) \text{ if } h_t(x) = f(x) \\ \exp(\alpha_t) \text{ if } h_t(x) \neq f(x) \end{cases}$

$$= \frac{\mathcal{D}_t(x) exp(-\alpha_t f(x) h_t(x))}{Z_t}$$

% Az eloszlás újraszámolása
% Z_t normalizáló tag,
% $\mathcal{D}_{t+1}(x)$ eloszlás lesz.

8. **end**

Kimenet:

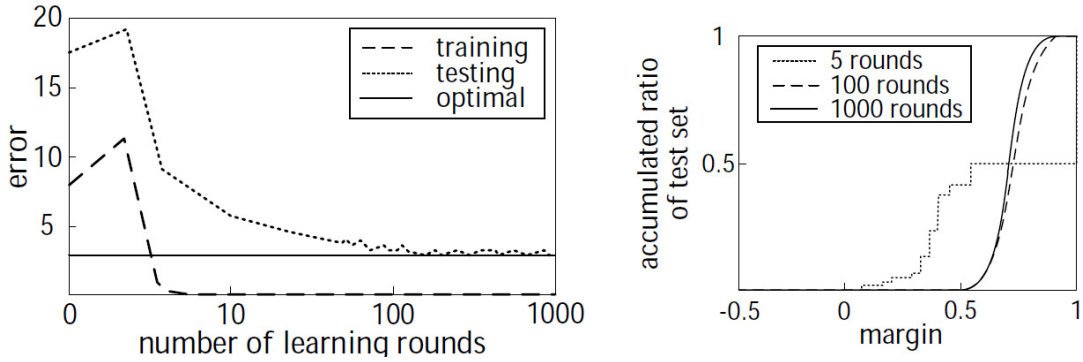
$H(x) = \text{sign} \left(\sum_{t=1}^{T} \alpha_t h_t(x) \right)$ % A tanulók additív $(h_1, \ldots, h_t)$ súlyozott kombinációja.

2.2. algoritmus. AdaBoost algoritmus

2.2.1. Megjegyzés. *Az AdaBoost algoritmus könnyebb megértéséhez az A.3. részben található példa nyújt segítséget. A mintafeladaton 10 adatpontra, ábrákon keresztül láthatók a lépések [19, 1.2.1 p. 7-10]. A feladathoz használt segédkódok az A.7. részben találhatók.*

Az AdaBoost algoritmus minden ciklusában ellenőrzi, hogy a kapott tanuló jobban teljesít-e, mint a $0R$ algoritmus, vagyis a véletlen tippelés. Ezt értelmességi kritériumnak

(ellenőrzésnek) nevezzük. Abban az esetben, amikor gyenge tanulóink száma jóval kevesebb, mint a T ciklusszám, ezt az értelmességi kritériumot megsértjük. Ennek nem teljesülése esetén korai befejezési probléma lép fel, ami gyakori a több osztályos feladatoknál [29, 2.2 p. 28]. Az AdaBoost algoritmussal kapcsolatban érdekes megfigyelések, felvetések merültek fel a matematikusokban, amelyeket máig nem tudtak bizonyítani, illetve elégséges feltételeket adni teljesülésükre. Ennek egyike, hogy mérések azt mutatják, hogy az AdaBoost algoritmus nem hajlamos a túltanulásra. Ez azt jelenti, hogy a teszt hiba akkor is csökken, amikor a tanító hiba már elérte a nullát. Schapire 1998-ban grafikonon szemléltette az AdaBoost tipikus tanító- és teszt hibáját [29, 2.4.2 p. 33].



2.1. ábra. Az AdaBoost tanító- és teszt hibájának szemléltetése

Ezt a jelenséget próbálta igazolni Schapire margók – tartóvektor gépeknél alkalmazott fogalom – alapján, amik bár befolyásolják a túltanulás mértékét, de nem magyarázzák meg teljesen. A margókkal kapcsolatos megfigyelés alapján fejlesztették tovább a margó minimalizáló *arc-gv* modellt *(Breiman, 1999)*. Ez a modell rosszabb teljesítményt ért el az AdaBoostnál, ezért a minimális margó ötletét elvetették. Az *arc-gv* modell alapján próbálkoztak azzal, hogy a teljesítmény javításához egy újabb modellbe a margók átlagát, vagy margók mediánját érdemes belevenni. A kutatások a témában máig folynak.

Érdemes foglalkozni az AdaBoost statisztikai megközelítésével, ami statisztikai fogalmakon keresztül magyarázza el az algoritmus működését. Ez nem teljesen azonos a fenti geometriai szemlélettel, de általában ezt használják. Az első eredményeket az AdaBoost statisztikai megközelítésénél Friedmann érte el, aki megmutatta, hogy az AdaBoost additív logisztikus regressziós modellek illesztésére való becsléssorozatként is értelmezhető. Friedmann levezetése alapján belátható, hogy maga az AdaBoost algoritmus felfogható egy optimalizáló eljárásnak, ami additív modelleket próbál illeszteni a helyettesítő

veszteségfüggvények alapján. A helyettesítő veszteségfüggvények *a 2.5.* fejezetben találhatók. Optimális esetben a helyettesítő veszteségfüggvény konziszetens [29, 2.4.3 p. 35-38], [9, 10.2-10.3 p. 341-343].

Az AdaBoost algoritmus különböző változataihoz – például MadaBoost, FilterBoost – a helyettesítő veszteségfüggvény különböző megválasztásával jutunk.

2.3. MadaBoost változat a túlsúlyozás mérséklésére

A 2.2. ábra az egyes elemek súlyozását, valamint az α, $w+$ és w- értékeket mutatja a tanulás kezdetén és az egyes ciklusok végén. A pirossal jelzett súlyok $w+$ szorzóval a feketén maradottak w- szorzóval jelennek meg a következő sorban.

2.3.1. Megjegyzés. *A táblázat eredményei az A.3 függelékben található feladat adatpontjai alapján számolhatók. A számításhoz használt programkódok az A.7 részben találhatók.*

	alfa	w+	w-	O (2,8)	O (4,3)	O (6,6)	O (7,8)	O (8,4)	X (1,6)	X (3,5)	X (3,7)	X (5,1)	X (7,2)
kezdeti				0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1
1.vágás után	0,424	1,528	0,655	0,071	0,071	0,071	0,071	0,071	0,167	0,167	0,167	0,071	0,071
2.vágás után	0,650	1,915	0,522	0,045	0,167	0,167	0,045	0,167	0,106	0,106	0,106	0,045	0,045
3.vágás után	0,923	2,517	0,397	0,167	0,096	0,096	0,026	0,096	0,061	0,061	0,061	0,167	0,167

2.2. ábra. súlyozások alakulása

Ha a Boosting folyamata sok cikluson át folyik, egyes elemek az egyre emelkedő $w+$ súlyok miatt aránytalanul túlsúlyozódhatnak. Több megoldási módszer született ennek a hibának a kiküszöbölésére, vagyis a túlságosan nagy súlyértékek – nagy $w+$ súlyok – csökkentésére. Ezek közül a *MadaBoost* működését mutatjuk be a probléma rövid elméleti megközelítése után.
A súlyok kezdeti eloszlása:

$$D_0 = \frac{1}{N}(1, 1, \ldots, 1)$$

A súlyok eloszlása az első ciklus után:

$$D_1(x) = \frac{D_0(x)}{Z_1} \exp(-\alpha_1 f(x) h_1(x))$$

A súlyok eloszlása a t-edik ciklus után:

$$D_t(x) = \frac{D_{t-1}(x)}{Z_1} \exp(-\alpha_1 f(x) h_1(x)$$

Ugyanez kifejtve:

$$D_t(x) = \frac{D_0(x)}{Z_t} \prod_{i=1}^{t} \exp(-\alpha_i f(x) h_i(x))$$

Az exp függvényt kihozva a szorzás elé:

$$D_t(x) = \frac{D_0(x)}{Z_t} \exp \sum_{i=1}^{t} -\alpha_i f(x) h_i(x) = \frac{D_0(x)}{Z_t} \exp(w_t^x)$$

$$\text{ahol } w_t^x = \sum_{i=1}^{t} -\alpha_i f(x) h_i(x)$$

A fentiek alapján az is egy lehetséges számítási mód, hogy az egyes $-\alpha_i f(x) h_i(x)$ értékeket tároljuk, majd az új w_t^x értéket ezek összegzésével nyerjük. Ezt az értéket exponenciális súlynak is nevezhetjük. A következő táblázatban a minták súlyai helyett ezek az exponenciális együtthatók láthatóak. Külön pirossal kiemelten (nyomtatásban halványabban szedve) találhatók a pozitív értékek, ezek a nagy súlyok.

A túlzottan nagy súlyértékek által okozott torzítások csillapítására a MadaBoost algoritmus az egyik legegyszerűbb módszer. Úgy küszöböli ki a túlzott súlyozást, hogy a pozitív együtthatókat egyszerűen 0-ra cseréli. Igaz, a következő ciklusnál nem az adott 0 értékkel számol tovább, hanem az eredeti pozitív értékkel, így ennek a könnyítésnek a hatása csökken az algoritmus végrehajtása során. További módosítás a következő lépésnél az AdaBoost eljárástól eltérő $\alpha, w+$ és $w-$ értékek számolása. A táblázat második adatcsoportjában a MadaBoost eljárás együtthatói szerepelnek.

	alfa	w+	w-	O (2,8)	O (4,3)	O (6,6)	O (7,8)	O (8,4)	X (1,6)	X (3,5)	X (3,7)	X (5,1)	X (7,2)
kezdeti				0	0	0	0	0	0	0	0	0	0
1.vágás után	0,424	1,528	0,655	-0,424	-0,424	-0,424	-0,424	-0,424	0,424	0,424	0,424	-0,424	-0,424
2.vágás után	0,650	1,915	0,522	-1,073	0,226	0,226	-1,073	0,226	-0,226	-0,226	-0,226	-1,073	-1,073
3.vágás után	0,923	2,517	0,397	-0,150	-0,697	-0,697	-1,996	-0,697	-1,149	-1,149	-1,149	-0,150	-0,150
	0,424	1,528	0,655	-0,424	-0,424	-0,424	-0,424	-0,424	0,000	0,000	0,000	-0,424	-0,424
	0,526	1,691	0,591	-0,949	0,000	0,000	-0,949	0,000	-0,102	-0,102	-0,102	-0,949	-0,949
	0,829	2,291	0,436	-0,120	-0,727	-0,727	-1,778	-0,727	-0,931	-0,931	-0,931	-0,120	-0,120

2.3. ábra. MadaBoost eljárás együtthatói

2.4. Gradient Boosting és XGBoost

Gradient Boosting algoritmus

A következőkben egy általános GBM *(Gradient Boosting Machine)* algoritmus lépéseit mutatjuk be. Érdemes megjegyezni, hogy az algoritmus során $h(x, \theta)$ alaptanuló típusa többféle lehet (például fa, neurális háló). A θ paraméterek egy adott függvényosztály paramétereinek felelnek meg. Ha h neurális háló, akkor a θ vektor elemei a háló elemeinek a súlyait jelentik. Ha h egy döntési fa, akkor θ azon vágási feltételeket írja le, amelyeket az adott fa jelent. A θ által kifeszített paramétertér minden egyes értelmezett pontja egy függvénynek felel meg [21]. Általában a Gradient Boosting algoritmusban az alaptanulók regressziós fák [17]. Ezért szokás GBM helyett GBRT-ként *(Gradient Boosted Regression Trees),* vagy MART-ként *(Multiple Additive Regression Trees)* is jelölni.

Gradient Boosting algoritmus

---

1. Bemenet:

(a) Legyen egy adott adathalmaz: $\{(x_i, y_i)\}_{i=1}^N$

(b) Választhatóak: M: interációs lépések száma

L: veszteségfüggvény

$h(x, \theta)$: alaptanuló függvények

(c) Kezdeti érték egy konstans:

$$f_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^N L(y_i, \gamma)$$

2. Folyamat:

for $m = 1$ to M :

(a) for $i = 1, 2, \ldots, N$ képezzük:

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$$

(b) Illesszünk egy regressziós fát az r_{im} célváltozó-reziduálisokhoz, külön minden $R_{jm},\ j = 1, 2, \ldots, J_m$ levélhez.

(c) for $j = 1, 2, \ldots, J_m$

$$\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$

(d) Számoljuk újra:

$$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$$

3. Kimenet:

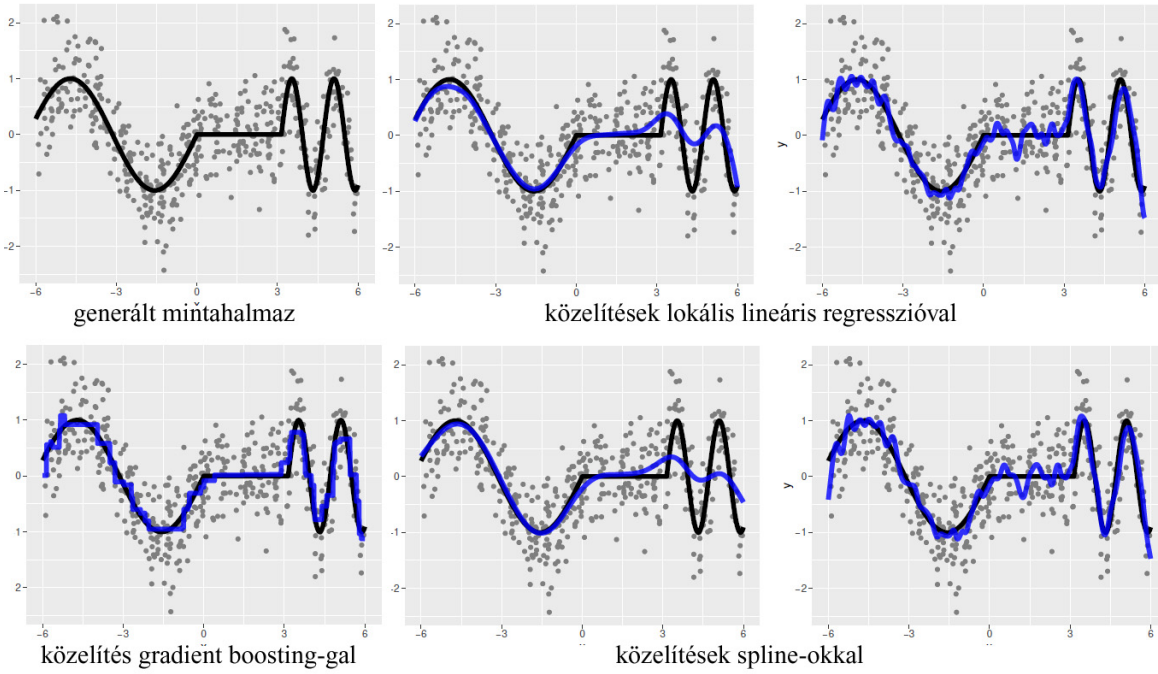
$f(x) = f_M(x)$

2.3. algoritmus. Gradient Boosting algoritmus

2.4.1. Megjegyzés. *Az Gradient Boosting algoritmus működését az A.4 részben található magyarázó példákban ismertetjük egyszerűbb, és bonyolultabb feladaton is.*

Miért teljesít kiválóan az XGBoost?

D. Nielsen az XGBoost jó teljesítő képességét a következő kísérleten demonstrálta [17, 9-10 p.79-92]. *A 2.4.* ábra bal oldalán fent látható szeszélyes függvényt generálta, majd a függvény értékeihez egy N(0, 0.5) zajt adott hozzá 500 pontban. Ez az 500 pont adja a mintatér pontjait, és a függvényérték a hozzáadott zajjal együtt lesz a célfüggvény. Azt a modellt keressük, amellyel a legjobban vissza tudjuk nyerni az eredeti zajmentes függvényt.



2.4. ábra. Nielsen különböző eljárásokkal közelíti a konstruált mintafüggvényt

Nielsen kísérletének eredményei *a 2.4.* ábrán láthatók. A felső sor bal oldalán csak a generált függvény, a továbbiakon azok közelítései is láthatók. A felső sorban még két különböző paraméterrel beállított lokális lineáris regresszióval közelíti a célfüggvényt. Az alacsony flexibilitású változat (középen) jól követi a $\sin(x)$ és a zéró konstans szakaszt is, de a $\sin(4x)$ szakaszon nagyon rosszul teljesít. A mellette jobbra látható változat nagyobb flexibilitású, ez meglepően jól követi a $\sin(4x)$ szakaszt, viszont főleg a zéró szakaszon gyorsan bemozdul a zajokra és elég nagy hibákat hoz össze. Ezen két diagram alatt az előbbiekhez nagyon hasonlóakat találunk, ezúttal spline-os közelítésekkel.

Az eredeti mintahalmaz alatt látható Nielsen közelítése additív három csúcsú (bináris) döntési fás Gradient Boostinggal. A ciklusszám $T = 4000$, az iterációs sebesség $\upsilon = 0,02$

és minden szelektor egyszerű három csúcsú döntési fa. A közelítések közül láthatóan ez a legjobb. Ezzel Nielsen megmutatta, hogy a Gradient Boosting kiválóan teljesít, de adós maradt a válasszal a miértre.

Az XGBoost hasznosíthatóságát az is mutatja, hogy 2015-ben a Kaggle-ben közzétett 29 verseny nyertesei közül 17 megoldás használt XGBoostot [1]. Ezen megoldások között 8 kizárólag XGBoostot használt a modell tanítására, míg a többi az XGBoost-ot kombinálta a neurális hálókkal, ami maga is a második legnépszerűbb módszer, melyet 11 megoldásban használtak. A KDD Cup 2015-ben is meggyőző volt, ahol az XGBoostot minden győztes csapat használta a top 10-ben.

Chen szerint az XGBoost előnyei az alábbiak [1]:

- Jól skálázható fa alapú rendszer.
- Newton módszer a gradiens módszer helyett a gyorsabb konvergencia érdekében.
- Elméletileg igazolt súlyozott kvantilis váz a hatékonyabb arányos kalkulációkhoz.
- Párhuzamos fa alapú tanulás egy új *sparsity-aware* algoritmussal.
- Hatásos cache memória blokk struktúra a külső memóriában tárolt fa tanulásához.

A felsorolásból kitűnik, hogy az XGBoost más, mint az eddigi algoritmusok, bár az alkalmazott új elemek nagy része korábban is létezett. Maga T. Chen említi például, hogy a Newton módszert még Friedman javasolta 2001-ben. Valószínűleg az akkori számítógépeknek a túl bonyolult számítások miatt inkább csak hátrány lett volna, összességében nőtt volna a futási idő. Ma már más a helyzet a számítástechnika dinamikus fejlődése miatt. A többi pontban említett eszközök is inkább informatikai, mint statisztikai jellegűek, amik növelik az eljárás teljesítményét. Ezen megoldások összessége tette az XGBoost-ot a ma ismert leghatékonyabb eljárássá.

2.5. Veszteség- és kockázatfüggvények

A veszteségfüggvény kiválasztása a közelítő eljárások lényeges pontja. A veszteségfüggvény határozza meg a közelítés irányát, sebességét. A Boosting algoritmusnál ez alapján képezzük a kockázatfüggvényt, aminek minimumát keressük az eljárás során.

2.5.1. Megjegyzés. *A fejezet a [16] forrás alapján készült. Jelölésrendszerében is jellemzően ezt a forrást követi.*

Eljárás:

1. Az y célváltozó egy lehetséges becslése: $\hat{y} = f(x)$
2. A veszteségfüggvény: $L(y, \hat{y}) = L(y, f)$
3. A kockázatfüggvény: $R(H) = \mathbb{E}_{X.D}(L(y, f))$
4. Végül a becslés optimalizálása: $f_X(x) = \underset{f}{\operatorname{argmin}}\, \mathbb{E}_{X.D}(L(y, f))$

Veszteségfüggvények fajtái:

A klasszikus valószínűségszámításban az alábbi három veszteségfüggvény közismert (természetesen a mai modern statisztikában már használnak mást is):

1. L_2 veszteségfüggvény: $L_2 = (y - f)^2$
2. L_1 veszteségfüggvény: $L_1 = |y - f|$
3. L_0 veszteségfüggvény: $L_0 = \mathbb{I}(y \neq f)$

2.5.2. Megjegyzés.

$$L_0 = \begin{cases} 0, & ha\ y = x \\ 1, & \forall\ x \in ET \mid y \neq x \end{cases}$$

ahol ET az értelmezési tartomány.
Vagyis az L_0 veszteségfüggvény a hibák darabszámát állapítja meg. Itt csak kétféle távolság létezik: a közel *és a* távol.

Melyik veszteségfüggvény használata optimális?

Az L_2 veszteség választása a leggyakoribb, mivel tapasztalati tény (Boosting), hogy az iteráció sokkal gyorsabb, mint az L_1 veszteség esetén. Ugyanakkor az L_1 veszteség sokkal stabilabb, mint az L_2. A négyzetes hiba használatának az a problémája, hogy csak a közel normális eloszlású adathalmazok esetén becsül hatékonyan az L_2, ettől távolodva a kiugró értékekre érzékeny. Ilyen esetekben az L_1 veszteség minimalizásával a célváltozó értékére kapott becslés (jóslat) sokkal megbízhatóbb. Mivel L_1-nek nem értelmezhető a deriváltja a 0 pontban, ez nehezíti a használatát, de mivel sokkal stabilabb, mint az L_2, szívesen alkalmazzák így is. A szakirodalomban találkozhatunk a *nyúl (L_2)* illetve

teknős (L_1) elnevezéssel is, célozva az iteráció sebességének jelentős különbségére *(Gauss-nyúl és Laplace-teknős)*. A két veszteségfüggvény előnyeit egyesítve javasolt Huber egy új helyettesítő függvényt, a *Huber veszteségfüggvényt* [11].

Huber veszteségfüggvény:

2.5.3. Definíció.

$$L_{Hub} = \begin{cases} \frac{L_2}{2}, & ha\ x \in [-\delta, +\delta] \\ L_1, & egyébként \end{cases}$$

A Huber veszteség meghatározása:

1. $L(y, f) = \frac{1}{2}(y - f)^2$, ha $|y - f| < \delta$
2. $L_{Hub} = \delta(|y - f| - \frac{\delta}{2})$, ha $|y - f| < \delta$

Huberhez hasonlóan Tukey is előállított egy veszteségfüggvényt.

Tukey veszteség:

A Tukey veszteségfüggvény az L_2 veszteséghez hasonlóan viselkedik a $[-\delta, +\delta]$ tartomány egy részében. Viszont a tartományhatárok közelében már inkább L_1, majd azon kívül az előbb felvetett esetleges L_0 veszteséghez hasonlóan viselkedik, vagyis állandó értéket ad. A függvény általános megnevezése: *kétsúlyú Tukey hibafüggvény.*

2.5.4. Megjegyzés. *Tukey veszteségnek csak a deriváltja zárt alakú:*

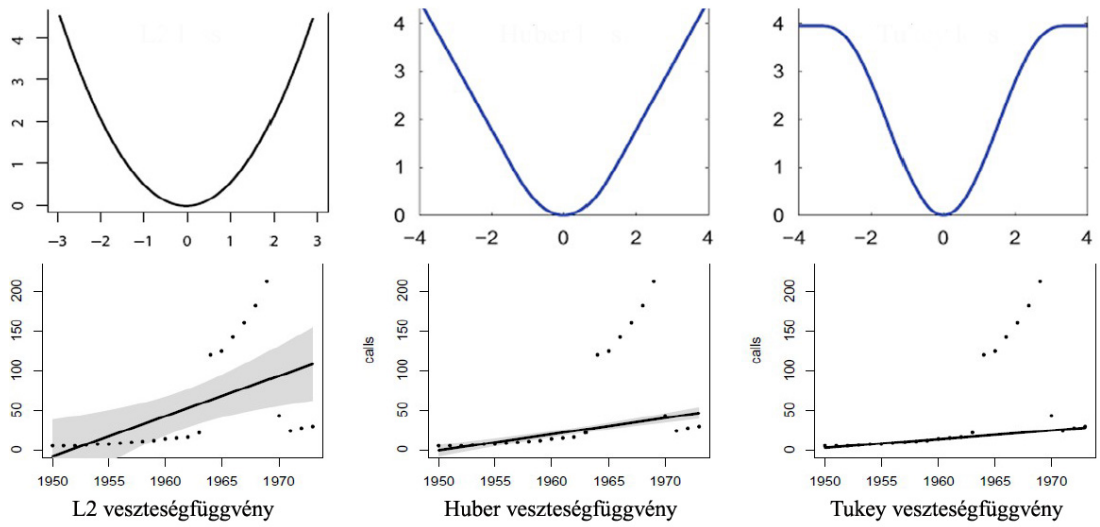
$$L'_{Tuk} = (y - f)\left(1 - \left(\frac{y - f}{\delta}\right)^2\right)^2, \ ha\ |y - f| < \delta,\ egyébként\ 0.$$

2.5.5. Gyakorlati példa. *A 2.5 ábra a tiszta L_2 veszteséget, és az előbbi két helyettesítő függvény képét, majd alatta azok hatását mutatja be olyan statisztikán, amely nagyon kevert eloszlásúnak látszik. A statisztika a belga telefontársaság hívásainak számát mutatja éves bontásban, ami szerint érthetetlen módon a belgák néhány egymást követő évben kiugróan sokat telefonáltak a többi évhez képest. A furcsa jelenség oka, hogy a belga telefontársaság az éves hívásmennyiség darabszáma helyett az 1964-70 közötti időszakban történt hívás időtartamát számolta percben, így természetes, hogy sokat nőttek az értékek a* mértékegység különbség *miatt.*

2.5.6. Modellezési megoldás. *Megfigyelés: Ezek a megsokszorozódott értékek egy külön második eloszlást alkotnak, így két dimenziós (de lehet több is, csak kettő látszik szemmel) eloszlásúnak tekinthetjük az adatszetet, melynek értékeit szeretnénk minél pontosabban közelíteni általános esetben. Erre a kihívásra az L_2 veszteség segítségével generált regressziós egyenes nagyon előnytelen becslést ad, mivel semmitmondó tartományban mozog. Sokkal jobban becsül a Huber-veszteség, de nem meglepő módon ebben az esetben a Tukey-veszteség teljesít a legjobban – mivel erre az alapfeladatra alkotta meg Tukey. A Tukey-veszteség segítségével képzett regressziós egyenes végig az alsó értékek mentén halad, jól mutatva a hívások darabszáma szerinti trendet.*

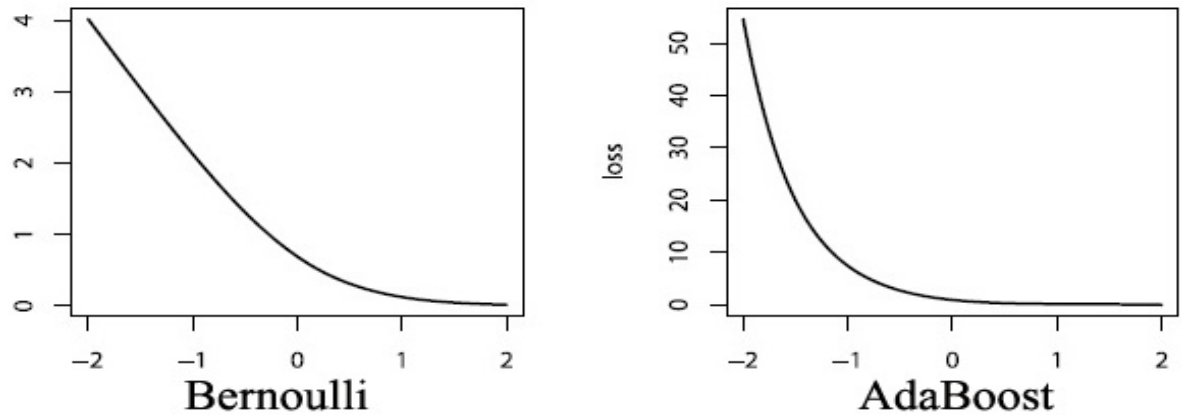
2.5.7. Megjegyzés. *Ha a Tukey regressziós egyeneshez a $[-\delta, +\delta]$ tartományban illeszkedő pontokat kivennénk az eloszlásból, és ezekre külön illesztenénk görbét a Tukey-veszteség segítségével, akkor a lenti ábrán látható felső értékekhez jól illeszkedő újabb egyenest kapnánk. Ez értelmes direkt eljárás lehet egy kevert eloszlásbeli minta modellezésére, célváltozójának jóslására.*

2.5.8. Megjegyzés. *A szakirodalomban a Tukey veszteség iterációs sebességére, konvergenciájára nem történt utalás (mivel akadályozza az iterációt a kezdeti L_0 loss felvétele miatt, ott tolonganak az adatok). Ezért Boosting modellekbe nagy adathalmazok esetén nem érdemes választani.*



2.5. ábra. A Huber- és Tukey- veszteségfüggvény összehasonlítása

Loss függvények osztályozás esetén



2.6. ábra. Kategorikus lossok

Kategorikus válasz esetén az y célváltozó általában $y \in \{0, 1\}$ bináris értékeket vesz fel. Tételezzük fel, hogy ez egy Bernoulli-eloszlásból származik. A jelölés egyszerűsítéséhez alkalmazzuk a transzformált y címkéket, melyre $y = 2y - 1$ és ekkor $y \in \{-1, 1\}$. Ebben az esetben az osztályválasz az új osztálycímkékhez kapcsolódó negatív log-likelihood minimalizálásával becsülhető: $L_{Bern}(\hat{y}, f) = \log\left(1 + \exp(-2\hat{y}f)\right)$. Ezt a veszteségfüggvényt Bernoulli veszteségnek nevezik. A Bernoulli-loss *a 2.6.* ábra bal oldalán látható. A diagram az yf értékek fölött definiált veszteségfüggvényt mutatja. Vegyük figyelembe, hogy ebben a jelölésben az yf pozitív értékei megfelelnek a helyes osztályzásnak.

A kategorikus veszteségfüggvény másik közös választása az egyszerű exponenciális veszteség, amivel az AdaBoost algoritmusban (Schapire, 2002) is találkozunk: $L_{Ada}(\hat{y}, f) = \exp(-\hat{y}f)$. Ez a függvény *a 2.6.* ábra jobb oldalán látható. További lehetőség a hinge-loss: $L_{hinge}(\hat{y}, f) = \max(0, (1 - \hat{y}f))$. Ez a loss-függvény *SVM-loss* (tartóvektorgép-veszteség) néven is ismert, ami népszerű felhasználási területére is utal.

2.6. Kitekintés további Boosting algoritmusokra

Annak ellenére, hogy az AdaBoost-ra vonatkozó margó alapú magyarázat *(15. oldal)* jó geometriai magyarázat a felhasználók többsége számára, a statisztikusok megpróbálták az AdaBoost algoritmust statisztikai módszerek felől közelíteni. Friedman (2000) megmutatta, hogy az AdaBoost algoritmust egy *stagewise* becslési eljárásként értelmezhetjük additív logisztikus regressziós modell illesztésére. Az AdaBoost.M1 algoritmus az exponenciális veszteségfüggvénnyel csak az egyik lehetséges megoldás, amelyet tipikusan a

rossz besorolási hibák mérésére használnak (Schapire és Singer, 1999). Ha logisztikus függvényt veszünk és megbecsüljük a valószínűséget, azt találjuk, hogy az exponenciális veszteségfüggvény és a logaritmikus veszteségfüggvény ugyanazt a függvényt minimalizálják. Friedman azt javasolta, hogy illeszkedjen a mintához egy logaritmikus regressziós modell úgy, hogy optimalizálja a logaritmikus veszteségfüggvényét gradiens segítségével. Ez vezetett el végül a LogitBoost algoritmushoz.

Friedman magyarázata szerint (2000) az AdaBoost egy olyan optimalizálási folyamat, amely egy helyettesítő veszteségfüggvényt alkalmaz. Ideális esetben a helyettesítő veszteségfüggvénynek konzisztensnek kell lennie, vagyis a helyettesítő veszteség optimalizálása megfeleltethető a veszteségfüggvény hibaarányai optimalizálásának. Ezek alapján az AdaBoost számos változatát úgy alakították ki, hogy különböző helyettesítő veszteségfüggvényeket vettek figyelembe. Ilyen a LogitBoost, amely logaritmikus, vagy az L2Boost amely L_2 veszteségfüggvényt használ.

LogitBoost algoritmus

Bemenet:

Adathalmaz: $D = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$;
Legkisebb négyzetes alap tanuló algoritmus: $\mathcal{L}$;
Tanulási folyamat ciklusszáma: T.

Folyamat:

1. $y_0(x) = f(x)$. % Célváltozó inicializálása
2. $H_0(x) = 0$. % Függvény inicializálása
3. **for** $t = 1, \ldots, T$:
4. $p_t(x) = \frac{1}{1+e\exp(-2H_{t-1}(x))}$; % Valószínűség kiszámítása
5. $y_t(x) = \frac{y_{t-1}(x)-p_t(x)}{p_t(x)(1-p_t(x))}$; % Célváltozó újraszámítása
6. $\mathcal{D}_t(x) = p_t(x)(1 - p_t(x))$; % Súlyok újraszámítása
7. $h_t = \mathcal{L}(D, y_t, \mathcal{D}_t)$; % A h_t legkisebb négyzetes klasszifikátor tanítása a $\mathcal{D}_t$-n alapuló D eloszlású adathalmazon úgy, hogy az y_t célváltozó értékét minél pontosabban meghatározza
8. $H_t(x) = H_{t-1}(x) + \frac{1}{2}h_t(x)$; % Kombinált klasszifikátorok újraszámítása
9. **end**

Kimenet:

$H(x) = \text{sign}\left(\sum_{t=1}^{T} h_t(x)\right)$ % A tanulók $(h_1, \ldots, h_t)$ lineáris kombinációja.

2.4. algoritmus. LogitBoost algoritmus

LPBoost algoritmus

Bemenet:

Adathalmaz: $D = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$;

Tanuló algoritmus: $\mathcal{L}$;

Paraméter: ν.

Tanulási folyamat ciklusszáma: T.

Folyamat:

1. $\omega_{1,i} = \frac{1}{m} \quad (\forall\ i = 1, \ldots, m)$
2. $\beta_1 = 0.$
3. **for** $t = 1, \ldots, T$:
4. $h_t = \mathcal{L}(D, w)$; % h_t tanuló tanítása az ω eloszlású D adathalmazból
5. **if** $\sum_{i=1}^{m} \omega_{t,i} y_i h_t(x_i) \leqslant \beta_t$ then $T = t - 1$; break; % optimalitás ellenőrzése
6. $H_{i,t} = h_t(x_i)(i = 1, \ldots, m)$ % Egy oszlop feltöltése
7. $(\omega_{t+1}, \beta_{t+1}) = \underset{\omega,\beta}{\operatorname{argmin}}\, \beta$

$$\text{s.t.} \quad \sum_{i=1}^{m} \omega_i y_i h_j(x_i) \leqslant \beta\ (\forall\ j \leqslant t)$$
$$\sum_{i=1}^{m} \omega_i = 1$$
$$\omega_i \in [0, \tfrac{1}{m\nu}] \quad (\forall\ i = 1, \ldots, m)$$

8. **end**
9. α kiszámítása $(\omega_{T+1}, \beta_{T+1})$ kettős megoldásból

Kimenet:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

% A tanulók $(h_1, \ldots, h_t)$ additív súlyozott kombinációja.

2.5. algoritmus. LPBoost algoritmus

3. fejezet

Bagging

A Bagging bemutatása előtt érdemes megjegyezni, hogy a hibrid modellek nem csak homogenitás (homogén, inhomogén) és felügyeltség alapján osztályozhatók, hanem a tanulók tanításának sorrendje alapján is. Nevezetesen vannak szekvenciális – például a Boosting algoritmusok az XGBoost kivételével – és párhuzamos hibrid modellek – például Bagging, XGBoost –. Szekvenciális modelleken olyan modelleket értünk, ahol a gyenge tanulókat egymás után tanítjuk. Ezzel szemben párhuzamos modelleknél a gyenge tanulók tanítása egyszerre, vagyis párhuzamosan történik. A párhuzamos modellek alapja, hogy független gyenge modellek kombinálása esetén a általános hiba jelentősen csökkenthető (exponenciálisan a gyenge tanulók száma szerint) [29, 3.1 p. 47-48]. A gyakorlatban ritkán kaphatók teljesen független alaptanulók, mivel azonos tanítóhalmaz alapján képeztük őket. Kevésbé összefüggő gyenge tanulókat véletlenítéssel kaphatunk. A párhuzamos hibrid módszerek további előnye, hogy a tanítási sebesség jelentősen csökkenthető több magos processzorokkal. A párhuzamos modellek közül a Bagging algoritmus kerül tárgyalásra. Ezt alkalmazzák, fejlesztik tovább a véletlen erdők különböző változatainál.

A Bagging egy *bootstap*-en és aggregáláson (összegzésen) alapuló módszer. Neve is innen ered: *Bootstrap AGGregatING* (Breiman, 1996d). Független tanulók kombinálása esetén a hibák jelentősen csökkennek, ezért a gyenge tanulók közti összefüggést szeretnénk minimálisra csökkenteni. Egy lehetséges módszer, hogy a tanítóhalmazból veszünk $k \in \mathbb{N}$ darab független részhalmazt, majd ezekre építünk külön-külön modellt. Általános esetben a tanítóhalmaz mérete relatíve kicsi, így ez a módszer kicsi, nem reprezentatív részhalmazokat gyárt. Ezeken a részhalmazokon tanított alaptanulók teljesítménye általában alacsony. Emiatt fejlesztették ki a Bagging algoritmust. A módszer *bootstrap* mintavételezéssel (Efron and Tibshirani, 1993) állítja elő az adathalmaz részhalmazait, amiken az alaptanulókat tanítjuk. Vagyis egy m elemű adathalmaz esetén m-szer mintavételezünk visszatevésesen (így van ismételt és kimaradt elem is), majd ezt a mintavételezést T-szer

végrehajtjuk. Így T darab m elemű mintánk van, amin külön-külön tanítjuk a gyenge tanulókat. Ezután klasszifikációs esetben szavazással, regressziónál átlagolással kombinálunk. A célváltozó értékét klasszifikációs esetben az alaptanulók címkéinek (például relatív többségi) szavazásával kapjuk. Érdemes megjegyezni, hogy a Bagging bináris és több osztályú adathalmazra ugyanúgy működik.

Bagging algoritmus

Bemenet:

Adathalmaz: $D = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$;

Tanuló algoritmus: $\mathcal{L}$;

Tanulási folyamat ciklusszáma: T.

Folyamat:

1. **for** $t = 1, \ldots, T$:
2. $h_t = \mathcal{L}(D, \mathcal{D}_{bs})$; % $\mathcal{D}_{bs}$ a bootstrap eloszlást jelöli
3. **end**

Kimenet:

$$H(x) = \operatorname*{argmax}_{y \in \mathcal{Y}} \sum_{t=1}^{T} \Pi(h_t(x) = y)$$

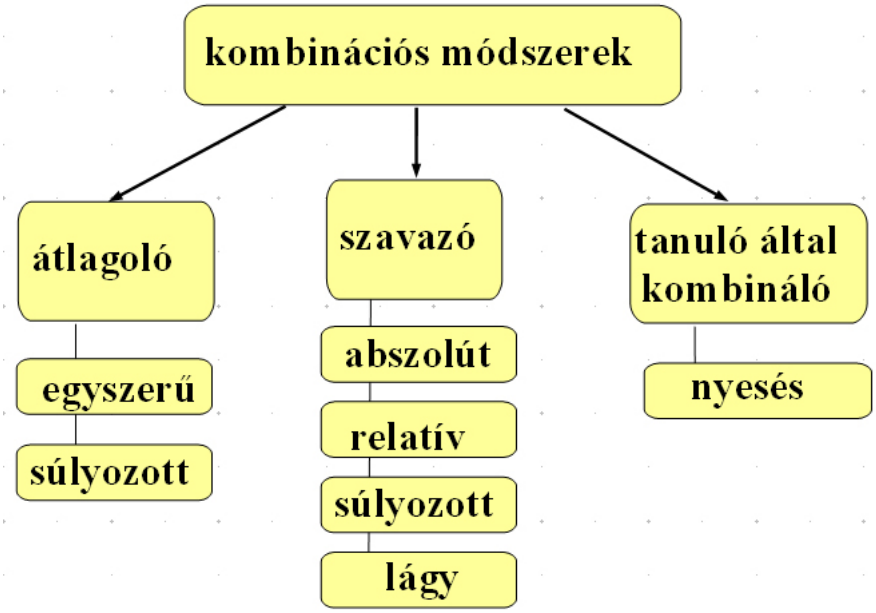
3.1. algoritmus. Bagging algoritmus

4. fejezet

Kombinációs módszerek

4.1. Kombinációs módszerek általános bemutatása

A hibrid modellek esetén a kombinációs módszerek fontos szerepet játszanak, mivel nem egy darab jól teljesítő tanulón, hanem több könnyen tanítható alaptanuló kombinálásán alapul a hibrid modell.



4.1. ábra. A kombinációs módszerek fajtái

Dietterich szerint kombinálással csökkenthetők az alábbi problémák [9, 10.2-10.3 p. 341-343]:

Statisztikai probléma: Statisztikailag, ha a hipotézistér túl nagy a tanító adathalmazhoz képest, és sok különböző tanulóval hasonló eredményre jutunk, nem lehetünk biztosak, melyik tanuló becsüli jól az y célváltozó értékét a tesztadatokon is. Ha kombináljuk a hipotéziseket (tanulókat), csökken a rossz hipotézis választásának valószínűsége, a hiba valamilyen szempontból átlagolódik.

Számítási idő probléma: A számítási időt csökkenti, mivel a gyenge tanuló algoritmusok jellemzően lokális optimum keresést hajtanak végre, így megakadhatnak egy lokális optimumban a globális helyett. A lokális optimum, vagyis a becsült érték gyakran nem jól közelíti a kívánt hipotézist. Ezzel szemben a különböző pontokból indított (szélsőértéket) kereső eljárások kombinációja általában jobban közelíti az optilmalizálandó modellt.

Közelítési probléma: A kombinálással kapott közelítés gyakran jobb becslést ad valós hipotézisre, mint a gyenge tanulók terének bármely eleme.

Azoknál a feladatoknál, ahol nagy a szórás, a számítási idő szórása vagy a jósolt érték és a valós hipotézis értéke messze van egymástól, ott nagy a torzítás, a hagyományos adatbányászati megközelítési módok nem megfelelőek.

4.1.1. Megjegyzés. *Ilyen esetekben – nagy szórás, nagy torzítás, nagy a számítási idő szórása – a hagyományos modelleknél jobban teljesítenek a hibrid modellek, mivel igazolták, hogy kombinációval a szórás és a torzítás is mérsékelhető a modelleknél.*

4.2. Átlag

Az átlag az egyik legelterjedtebb, egyszerű és kevés számítást igénylő kombinálási módszer. Ebben a fejezetben a regresszión keresztül mutatjuk be az átlagolás működési elvét.

A feladat:

Bemenet:

T darab tanuló: $(h_1, \ldots, h_T)$,
x (minta vektor)

Cél:

$(h_1, \ldots, h_T)$-t szeretnénk kombinálni úgy, hogy a hipotézisünk által adott közelítő értékeket kapjunk y-ra.

Kimenet:

$(h_1(x), \ldots, h_T(x)) \in \mathbb{R}$

4.1. algoritmus. Átlag

4.2.1. Egyszerű átlag

Az egyszerű átlag a különböző tanulók átlagát veszi az alábbi képlet szerint:

$$H(x) = \frac{1}{T} \sum_{i=1}^{T} h_i(x)$$

Az egyszerű átlag hátránya, hogy érzékeny a kiugró értékekre, zajokra, adathibákra, így szélsőséges adatok esetén nem javasolt az alkalmazása.

4.2.2. Súlyozott átlag

A tanulókat a célváltozó meghatározásában játszott szerepük alapján súlyozzuk (súlyok összege 1), majd ezeket az új értékeket átlagoljuk. Emiatt ez a módszer sokkal kevésbé érzékeny a kiugró értékekre [29], mivel a kevéssé szignifikáns változók súlyát minimálisra csökkentjük.

4.2.1. Megjegyzés. *Az egyszerű átlag ennek speciális esete, ahol minden súly azonos.*

Kiszámításának módja:

$$H(x) = \sum_{i=1}^{T} w_i h_i(x), \text{ ahol } w_i \geqslant 0 \text{ és } \sum_{i=1}^{T} w_i = 1$$

Kísérletek igazolják, hogy a súlyozott átlag nem mindig teljesít jobban az egyszerű átlagnál [29]. Mivel a becsült súlyok az adatok zajossága miatt gyakran megbízhatatlanok, így ez a módszer hajlamos a túltanulásra. Összességében az egyszerű átlagot főként hasonló teljesítményű tanulók kombinálására, míg a (nem azonosan) súlyozott átlagot a lényegesen különböző teljesítményű tanulók kombinálására használják.

4.2.2. Megjegyzés. *A súlyok optimális megválasztása nehéz probléma, így azt csak közelíteni tudjuk:*

$$w = \operatorname*{argmin}_{w} \sum_{i=1}^{T} \sum_{j=1}^{T} w_i w_j C_{ij},$$

ahol C_{ij} meghatározása és levezetése a [29, 4.2.2 p. 70] forrásban található. Ez főként elméleti jelentőségű.

4.3. Szavazás

Diszkrét kimeneteknél a szavazás népszerű kombinálási módszer. Klasszifikációs feladaton mutatjuk be, de klaszterezés esetén is alkalmazható [29].

A feladat:

Bemenet:

T darab tanuló: $(h_1, \ldots, h_T)$,

Cél:

a célváltozó osztály címkéjét meghatározni – amely az l darab $(c_1, \ldots, c_l)$ osztálycímke egyike – $(h_1, \ldots, h_T)$ segítségével.

Kimenet:

$(h_i^1(x), \ldots, h_i^l(x))^T$, ahol $h_i^j(x)$ a h_i tanuló kimenete a c_j osztálycímkére (l dimenziós címke vektor).

4.2. algoritmus. Szavazás

4.3.1. Definíció. *Éles címke:*

$$h_i^j(x) \in \{0,1\} = \begin{cases} 1, & \textit{ha } h_i\, c_j\textit{-t jósolja osztálycímkének} \\ 0, & \textit{egyébként,} \end{cases}$$

ahol $i \in \{1, \ldots, T\}$, *és* $j \in \{1, \ldots, l\}$

4.3.2. Definíció. *Osztályvalószínűség:*

$$h_i^j(x) \in [0,1]$$

ami a posterior valószínűség: $P(c_j \mid x)$ *becslésének tekinthető.*

4.3.3. Megjegyzés. *Nem normalizált margót adó klasszifikátorok estén (például SVM; kalibrációs módszerek: Platt méretezés, izotóniás regresszió) a kimenetek valószínűséggé konvertálására alkalmazható [29]. Megfigyelhető, hogy míg a kasszifikátorok által becsült osztályvalószínűségek önmagukban gyengék, kombinációjukat véve felveszik a versenyt az éles címkén alapuló módszerekkel.*

4.3.1. Abszolút többségi szavazás

A legnépszerűbb szavazó algoritmus. Minden klasszifikátor egy címkét ad kimenetnek. A címkeösszegzések után kapott címkeosztályok (azonos címkéjűek ugyanabba az osztályba tartoznak) elemszáma alapján kapjuk meg y jósolt értékét. Ha az egyik címkeosztály a szavazatok több mint 50%-át kapta, a célváltozónak ezt az értéket jósoljuk. Ha ez semelyik címkeosztályra nem érvényesül, akkor az elutasítási mechanizmus lép érvénybe, a kombinált klasszifikátor (hibrid modell) nem ad eredményt.

Formálisan leírva:

$$H(x) = \begin{cases} c_j, & \text{ha } \sum_{i=1}^{T} h_i^j(x) > \frac{1}{2} \sum_{k=1}^{l} \sum_{i=1}^{T} h_i^k(x) \\ \text{elutasít (nem ad eredményt)}, & \text{egyébként.} \end{cases}$$

Speciálisan bináris klasszifikáció esetén a hibrid modell által jósolt érték helyes, ha legalább $[\frac{T}{2}+1]$ tanuló azonos címkét választ. Tegyük fel, hogy a klasszifikátorok által jósolt értékek függetlenek egymástól. Ez gyakorlatban ritkán fordul elő, mivel a tanulók jellemzően valamennyire korrelálnak, hiszen azonos feladaton tanítjuk őket. Így ez a feltevés csak elméleti jelentőségű. Tegyük fel továbbá, hogy minden tanuló p valószínűséggel ad helyes választ. Ekor a hibrid modell által adott eredmény helyességét kiszámíthatjuk a binomiális eloszlás alapján. Minimum $[\frac{T}{2}+1]$ helyes klasszifikátorral rendelkezés valószínűségének kiszámítása T-ből:

$$P_{mv} = \sum_{k=[\frac{T}{2}+1]}^{T} \binom{T}{k} p^k (1-p)^{T-k}$$

4.3.4. Megjegyzés. *Különböző* p *valószínűségekre a kísérleti eredmények, megfigyelések a [29, 4.3.1 p. 72-73] forrásban találhatók.*

4.3.2. Relatív többségi szavazás

A relatív többségi szavazás alapgondolata azonos az abszolút többségi szavazáséval, de míg az előbbinél elfogadott, hogy elutasítjuk a kapott eredményt, itt a legtöbb szavazatot kapott címkét megkapja a célváltozó. Vagyis ha 100 osztály van ($l = 100$), és a legtöbb szavazatot kapott címkeosztály is csak 2%-át kapta a szavazatoknak, a célváltozó ezt az értéket kapja. Pedig nem valószínű, hogy a tényleges érték azonos az így jósolttal. Vagyis ez a módszer kevésbé törődik az eredmény megbízhatóságával.

$$\text{A módszer formálisan: } H(x) = \operatorname*{argmax}_j \sum_{i=1}^{T} h_i^j(x)$$

4.3.5. Megjegyzés. *Bináris esetben az abszolút és relatív többségi szavazás azonos. Nehezen megoldható feladat esetén sem kerülhet végtelen ciklusba (mivel általában egy elfogadható eredmény megtalálásáig próbálkozunk) az általános többségi szavazással ellentétben. Ezért számítási idő szempontjából jó választás lehet.*

4.3.3. Súlyozott szavazás

Különböző teljesítményű tanulók esetén ajánlott módszer. A nagyobb teljesítményű tanulók befolyását növeljük a szavazáskor, nagyobb súlyt adunk nekik, míg a kis teljesítményűekét csökkentjük.

A kimeneti osztály címke az alábbi képletből kapható meg:

$$H(x) = c_{\underset{j}{\operatorname{argmax}} \sum_{i=1}^{T} w_i h_i^j(x)}$$

ahol w_i a h_i klasszifikátorhoz tartozó súly.

4.3.6. Megjegyzés. *Gyakorlati megvalósításkor gyakran normalizált és pozitív súlyokkal dolgozunk, hasonlóan a súlyozott átlaghoz.*

Formálisan: $\sum_{i=1}^{T} w_i = 1,\ w_i \geqslant 0$

Megfelelő súlyozás választásával a súlyozott szavazás magasabb teljesítményű lehet a legjobb alaptanulónál és a többségi szavazás eredményénél. A súlyozott átlaghoz hasonlóan itt is az optimális súly megválasztása a kulcs. Szeretnénk megtalálni azt az optimális súlyozást, mely esetén a hibrid modell teljesítménye (ezen kombinációs módszer használatával) maximális, nem növelhető tovább. Erre a feladatra levezették az optimális megoldást független tanulók esetére [29, 4.3.2 p. 74-75]. A megoldást terjedelmi okokból levezetés nélkül közlöm.
Optimális súlyok: $w_i = log\frac{p_i}{1-p_i}$.

Ez az eredmény csak elméleti jelentőségű, jellemzően a függetlenség megsértése miatt.

4.3.4. Lágy szavazás

Az éles címkét eredményező tanulók esetén az abszolút és relatív többségi szavazás, továbbá a súlyozott szavazás alkalmazható. Ezzel szemben az olyan tanulók esetén, melyek

osztályvalószínűségeket adnak végeredményként, a lágy szavazás az általánosan elfogadott módszer.

Módszer: A h_i tanuló kimenete az x adatpontra (mintaelemre) az l dimenziós $(h_i^1(x), ..., h_i^l(x))^T$, ahol $h_i^j(x) \in [0,1]$ a $P(c_j \mid x)$ poszterior valószínűség becslésének tekinthető. Ha minden alaptanulót azonos befolyásúnak tekintünk az eredmény szempontjából, az egyszerű lágy szavazáshoz jutunk. Ha a tanulókat különböző súllyal vesszük be a kombinációba, a súlyozott lágy szavazást kapjuk, melynek több változata van.

Egyszerű lágy szavazás:

Az egyszerű lágy szavazás a tanulók kimeneteit átlagolja egyszerű átlag szerint. A c_j osztályhoz tartozó kimenet: $H^j(x) = \frac{1}{T}\sum_{i=1}^{T} h_i^j(x)$.

A súlyozott lágy szavazás változatai:

- Klasszifikátor alapú súlyozás: c_j osztályhoz tartozó kombinált kimenet $H^j(x) = \sum_{i=1}^{T} w_i h_i^j(x)$, ahol w_i a h_i klasszifikátorhoz tartozó súly.
- Osztály alapú súlyozás: c_j osztályhoz tartozó kombinált kimenet $H^j(x) = \sum_{i=1}^{T} w_i^j h_i^j(x)$, ahol w_i^j a c_j osztálybeli h_i klasszifikátorhoz tartozó súly.
- Mintaelem (adatpont) alapú súlyozás: c_j osztályhoz tartozó kombinált kimenet $H^j(x) = \sum_{i=1}^{T}\sum_{k=1}^{m} w_{ik}^j h_i^j(x)$, ahol w_{ik}^j a c_j osztálybeli h_i klasszifikátorhoz tartozó súly.

4.3.7. Megjegyzés. *Gyakorlati szempontból a mintaelem alapú súlyozott lágy szavazás kis jelentőségű, mivel a súlyok száma túl nagy, ezért nehezen kezelhető, nagy a számítási igény, és a súlyok optimális beállítása is nehézkes [29].*

4.3.8. Megjegyzés. *A klasszifikátor alapú súlyozott lágy szavazás nagyon hasonlít a súlyozott átlagra és sima súlyozott szavazásra.*

Osztály alapú súlyozott lágy szavazás esetén levezethető, hogy az optimális súlyok az alábbi optimalizálási feladat megoldásai:

$$H^j = \underset{w^j}{\operatorname{argmin}} \sum_{k=1}^{m} \left(\sum_{i=1}^{T} w_i^j h_i^j(x_k) - \mathbb{I}(f(x_k) = c_j) \right)^2, \; j = 1, ..., l.$$

[29, 4.3.4 p. 76-77]. A lágy szavazást homogén modellek esetén szokták alkalmazni. Heterogén modellek esetén általában csak akkor alkalmazható, ha a kimeneti osztályvalószínűségeket bináris osztálycímkékké konvertáljuk át:

$$h_i^j(x) = \begin{cases} 1 & \text{ha } h_i^j(x) = \max\limits_j h_i^j(x) \\ 0, & \text{egyébként,} \end{cases}$$

majd az éles címkékre használható szavazási módszereket alkalmazzuk.

4.4. Tanuló általi kombinálás

Tanulók általi kombinációs módszerek alatt olyan kétrétegű (kétszintű) modelleket értünk, melyeknél a kombinálási módszer maga is egy tanuló. Az alaptanulókat első szintű, míg a kombináláskor alkalmazott tanulót másodszintű tanulónak (meta tanulónak) nevezzük.

4.4.1. Megjegyzés. *Általánosabb esetben nem csak két, hanem $n \in \mathbb{N}$ szintből is állhat a modell.*

4.4.2. Megjegyzés. *Ha ezt a másodszintű tanulót indikátorként tekintjük, a nyesési eljárásokhoz jutunk, amiket általában a túltanulás kiküszöbölésére alkalmazunk.*

Tanulók általi kombinálás általánosan

Bemenet:

Adathalmaz: $D = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\ \}$;
Első szintű tanuló algoritmus: $\mathcal{L}_1, \ldots, \mathcal{L}_T$;
Második szintű tanuló algoritmus: $\mathcal{L}$;

Folyamat:

1. **for** $t = 1, \ldots, T$: % Első szintű tanuló tanítása az
2. $h_t = \mathcal{L}_t(D))$; % $\mathcal{L}_t$ Első szintű tanuló algoritmus segítségével.
3. **end**
4. $D' = 0$;
5. **for** $i = 1, \ldots, m$:
6. **for** $t = 1, \ldots, T$:
7. $z_{it} = h_t(x_i)$;
8. **end**
9. $D' = D' \cup ((z_{i1}, \ldots, z_{iT}), y_i)$;
10. **end**
11. $h' = \mathcal{L}(D')$; % A h' másodszintű tanuló tanítása a második szintű $\mathcal{L}$ tanuló algoritmus végrehajtásával az új D' adathalmazon.

Kimenet:

$H(x) = h'(h_1(x), \ldots, h_T(x))$ % A $(h_1, h_2, \ldots)$ első szintű tanulók kombinálása a h' másodszintű tanuló segítségével.

4.3. algoritmus. Tanulók általi kombinálás általánosan

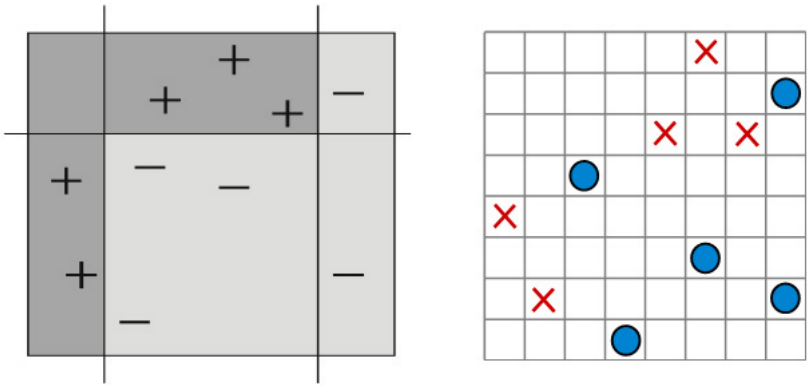
A. függelék

Szemléltető modellek, kódok

A.1. Táblás modell Boosting algoritmusokhoz

Bevezető a táblás modellhez

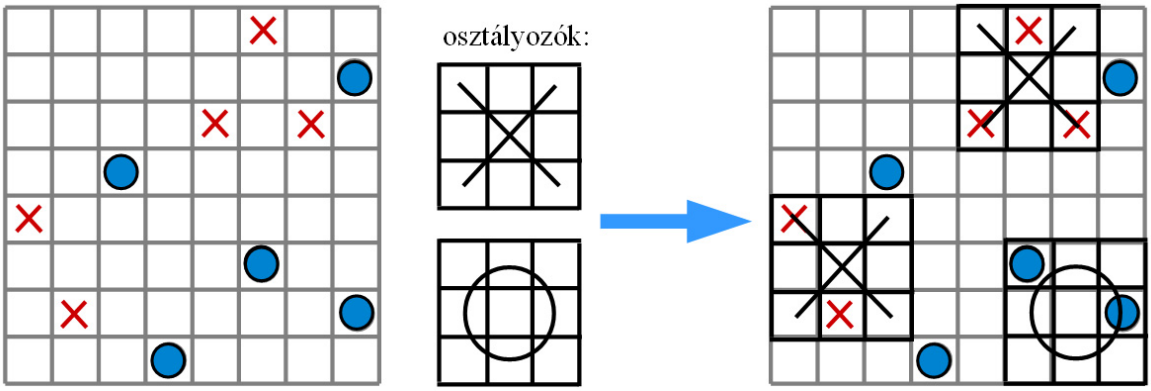
Schapire 2012-es könyvében [19] jelenik meg először az a feladat *(tiny toy learning problem)*, amely az AdaBoost algoritmus kedvező tulajdonságait mutatja be fejszámolással is követhető szinten. Azóta ez a kis játékfeladat sok AdaBoost ismertetés bevezetője lett, széleskörűen idézik az AdaBoost algoritmus bemutatására. A tiny toy feladat *az A.1.* ábrán látható. Egy meghatározatlan egyszerű majdnem négyzetben helyezkedik el 10 elem, amelyből 5-5 darab tartozik egy osztályba. Ott az egyes elemek pontos koordinátái nincsenek megadva. Schapire részletesen közli az eljárás mozzanatait, a súlyozás alakulását és egyéb jellemzőket. A saját táblás változat azzal a konkrét igénnyel készült, hogy ha úgyis kellenek valamilyen koordináták a modellezéshez, akkor az minél kvantáltabb legyen. Így a saját változat elemeit egy sakktáblán helyeztük el. Az AdaBoost.M1 eljárás szemléltetéséhez a vágások és a súlyozások ugyanazok az értékek maradtak.



A.1. ábra. A tiny toy és a táblás modell lényegében azonosak

Kis osztályozós játék

Ezen kis bevezető feladat előnye, hogy semmilyen matematikai apparátust nem igényel. A legrégebbi Boosting eljárások *többségi szavazás* elvét mutatja be. Tehát adott egy 8×8 -as táblán elrendezve 10 bábu (10 elem). Ezen elemeket osztályozni szeretnénk a rendelkezésre álló 3×3 -as kendőkkel, amelyekből több darab is létezik. Egy-egy kendő úgy osztályozza a bábukat, hogy azt a tábla egy 3×3 -as részére leterítve a kendő alattiakat tekinti a saját jelével azonosnak, az azon kívülieket pedig a másik osztályba sorolja. Minden kendő minden bábut besorol valamelyik osztályba, a végső besorolás pedig minden elemnél a többségi szavazás szabályai szerint dől el.



A.2. ábra. Több gyenge osztályozás összessége kiváló eredményt is adhat

Amikor ezt a feladványt valakinek feladtuk, a megoldó *az A.2.* ábra jobb oldalán látható helyzetig jutott el és ott elakadt. Nem vette észre, hogy már meg is oldotta a feladatot. Maradt három bábu, amelyre nem került kendő, de a megoldónak főleg azt kellett volna észrevennie, hogy ez nem is szükséges, hiszen már azok a bábuk is háromszor voltak

osztályozva. Kétszer helyesen O-nak a két X-jelű kendő által és egyszer helytelenül X-nek pont a saját O-jelű kendő által. Észrevehetjük, hogy miután a két X-jelű kendőt leterítettük, az O-jelűt bárhova teríthetjük, ami nem fed le X bábut. Vagyis például a kendőt a tábla bal felső sarkában elhelyezve, $\varepsilon = 0,5$ hiba mellett is sikeresen fejezhetjük be az osztályozást. Az egész feladat azt a tételt szemlélteti, hogy sok (vagy csak több) gyenge osztályozás összességében kiváló eredményt adhat.

A.1.1. Megjegyzés. *A könnyebb áttekinthetőség kedvéért* a 2.2. *fejezetben ismertetett algoritmus itt is szerepel.*

A.2. AdaBoost algoritmus

Bemenet:

Adathalmaz: $D = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$;
Alap tanuló algoritmus: $\mathcal{L}$;
Tanulási folyamat ciklusszáma: T.

Folyamat:

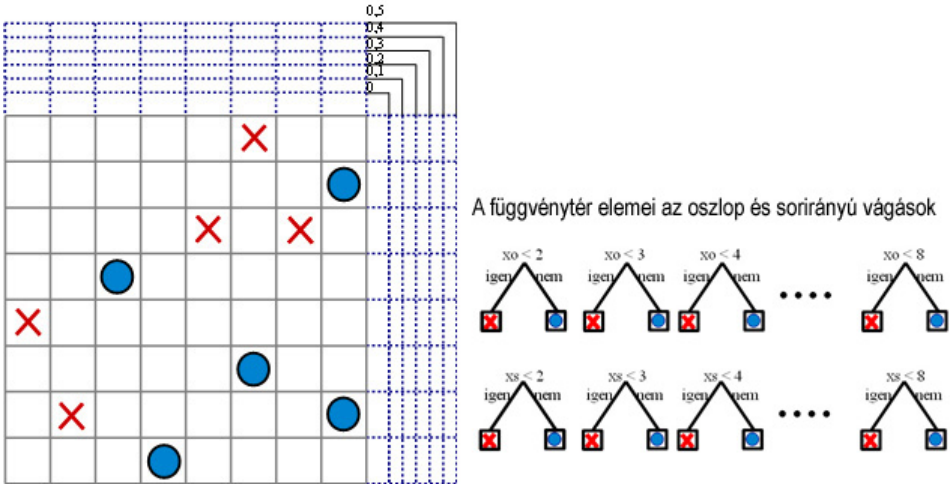
1. $\mathcal{D}_1(x) = \frac{1}{m}$. % Súlyok eloszlásának inicializálása
2. **for** $t = 1, \ldots, T$:
3. $h_t = \mathcal{L}(\mathcal{D}, \mathcal{D}_t)$; % h_t klasszifikátor tanítása a $\mathcal{D}_t$ eloszlású D alaphalmazon
4. $\varepsilon_t = P_{x \sim D_t}(h_t(x) \neq f(x))$; % h_t hibájának kiértékelése
5. **if** $\varepsilon_t > 0,5$ **then break**
6. $\alpha_t = \frac{1}{2} \ln(\frac{1-\varepsilon_t}{\varepsilon_t})$; % h_t tanuló súlyának meghatározása
7. $$\mathcal{D}_{t+1}(x) = \frac{\mathcal{D}_t(x)}{Z_t} \times \begin{cases} \exp(-\alpha_t) \text{ if } h_t(x) = f(x) \\ \exp(\alpha_t) \text{ if } h_t(x) \neq f(x) \end{cases}$$
$$= \frac{\mathcal{D}_t(x) exp(-\alpha_t f(x) h_t(x))}{Z_t}$$
% Az eloszlás újraszámolása
% Z_t normalizáló tag,
% $\mathcal{D}_{t+1}(x)$ eloszlás lesz.
8. **end**

Kimenet:

$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$ % A tanulók additív $(h_1, \ldots, h_t)$ súlyozott kombinációja.

A következő részben a táblás modell osztályozási feladat megoldását a fenti algoritmus lépései szerint mutatjuk be. A folyamat leírásánál a hivatkozás az algoritmus lépéseire az itt feltüntetett számokkal történik.

A.3. Táblás modell az AdaBoost algoritmussal



A.3. ábra. AdaBoost kezdeti lépések: adatok és függvények. Kezdeti súlyozás

1. Tanuló adatok: $D = \{(x_1, y_1), \ldots, (x_{10}, y_{10})\}$, ahol $y_1 = y_2 = \ldots, y_5 = -1$, és $y_6 = y_7 = \ldots, y_{10} = +1$, vagyis a (piros) X-ek 1, a (kék) O-k -1 értékűek.
   $x_i = (xs_i, xo_i,)$, ahol xs_i és xo_i az i-edik elem sor- és oszlopkoordinátái.

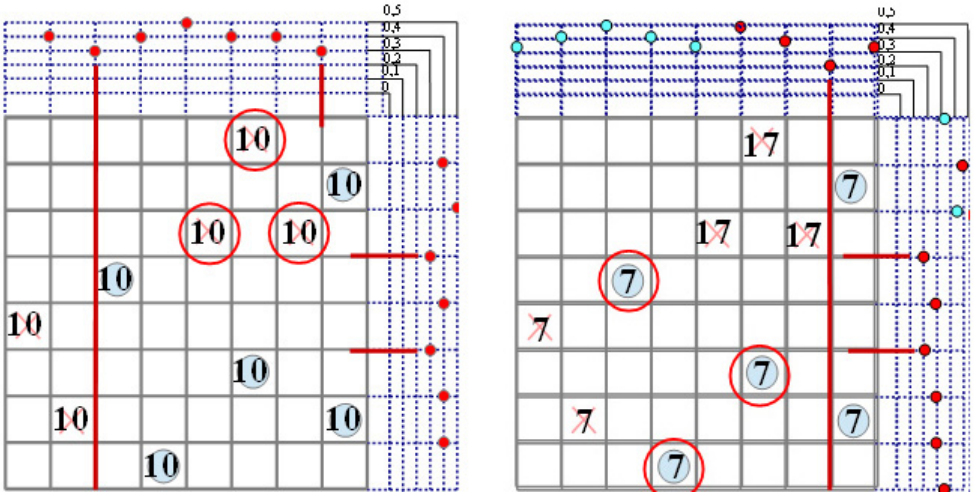
   $x_1 = (2, 8)$, $x_2 = (4, 3)$, $x_3 = (6, 6)$, $x_4 = (7, 8)$, $x_5 = (8, 4)$,
   $x_6 = (1, 6)$, $x_7 = (3, 5)$, $x_8 = (3, 7)$, $x_9 = (5, 1)$, $x_{10} = (7, 2)$

2. $D_1(1 : 10) = \frac{1}{10} = 0,1$. A további ábrákon a súlyozást százalékban tüntetjük fel, tehát a $0,1$ súly 10 értéken szerepel. Így természetesen a súlyok összértéke 100 lesz.

A gyenge tanulók legyenek a sorok és oszlopok menti vágások. A 8×8 -as méret miatt 7-7 függőleges illetve vízszintes vágás létezik. Ehhez a 14 vágáshoz tartozó hibaértékek olvashatók le az oszlopokhoz illetve sorokhoz illesztett diagramról balról jobbra, majd folytatva fentről le.

További lehetőség, hogy a vágás melyik oldalára osztályozzuk az X ill. O elemeket. Így összesen 4×7, azaz 28 döntési fát hozhatunk létre. Ez a 28 fa együttesen a döntési tér (függvénytér), amelyen az optimalizálást végezzük.

A.3.1. Megjegyzés. *Technikailag csak 14 fával foglalkozunk: 7 vízszintes és 7 függőleges fával, és figyeljük a hiba minimumát és maximumát egyaránt. A hiba maximumánál a komplementer vágásnak minimuma van.*



A.4. ábra. Az AdaBoost eljárás első két ciklusa. Vágások és átsúlyozások

3. $\mathbf{t = 1}$ ciklus (*az A.4.* ábra bal oldalán)

4. Legjobb vágások: függőlegesen a 2. oszlopnál, vagy a 7. oszlopnál. Vízszintesen a 3. sornál vagy az 5. sornál. Legelső kiválasztott a 2. oszlop.

5. $\varepsilon_1 = P_{(x,y)}[h_1(x) \neq y] = 0,3$

6. $\alpha_1 = \frac{1}{2} \ln \left(\frac{1-\varepsilon_1}{\varepsilon_1} \right) = 0,42$

7. Mivel $y_i h_1(i)$ szorzat csak +1 és -1 lehet, ezért csak kétféle érték fordulhat elő: $D_2(1:5) = D_2(9:10) = 0,07$ és $D_2(6:8) = 0,17$. Az átsúlyozás utáni állapot *az A.4.* ábra jobb oldalán látszik.

---

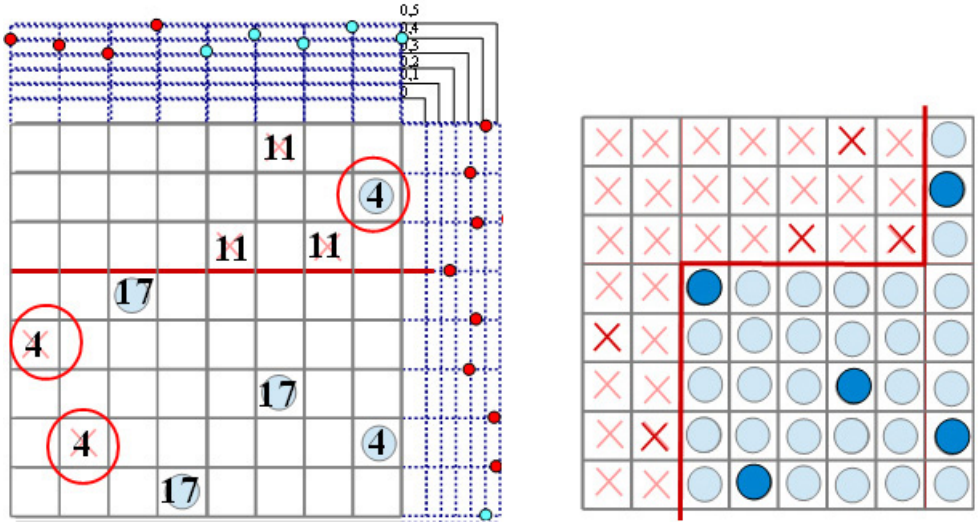
3. $\mathbf{t = 2}$ ciklus (*az A.4.* ábra jobb oldalán)

4. Legjobb vágások: függőlegesen a 7. oszlopnál. Vízszintesen a 3. sornál vagy az 5. sornál. Legelső kiválasztott a 7. oszlop.

5. $\varepsilon_2 = P_{(x,y)}\left[h_2(x) \neq y\right] = 0,21$

6. $\alpha_2 = \frac{1}{2} \ln \left(\frac{1-\varepsilon_2}{\varepsilon_2} \right) = 0,65$

7. Mivel $y_i h_2(i)$ szorzat ismét csak $+1$ és -1 lehet, ezért a négyféle értékből csak háromféle fordulhat elő: $D_3(1,4,9,10) = 0{,}045, D_3(2,3,5) = 0{,}167$, és $D_3(6:8) = 0{,}106$. Az átsúlyozás utáni állapot *az A.5.* ábra baloldali részén látszik.



A.5. ábra. Az AdaBoost eljárás harmadik ciklusa és a végeredmény

---

3. $\mathbf{t = 3}$ ciklus (*az A.5.* ábra bal oldala szerint)

4. Legjobb vágás: Vízszintesen a 3. sornál.

5. $\varepsilon_3 = P_{(x,y)}\left[h_3(x) \neq y\right] = 0{,}16$

6. $\alpha_3 = \frac{1}{2}\ln\left(\frac{1-\varepsilon_3}{\varepsilon_3}\right) = 0{,}92$

7. Mivel $y_i h_3(i)$ szorzat ismét csak +1 és -1 lehet, most már négyféle súly fordulhat elő: $D_4(1,9,10) = 0{,}087, D_4(2,3,5) = 0{,}125, D_4(4) = 0{,}009$, és $D_4(6,7,8) = 0{,}125$. Ezt a súlyozást már nem mutatjuk, mert a modell készen van. Ez *az A.5.* ábra jobb oldalán látható.

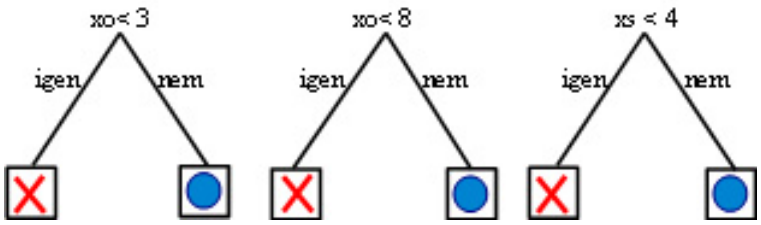
A megoldás a három döntési fa kombinációja:

$$y_i = sign\left(\sum_j \alpha_j f_j(x_i)\right)$$

Ellenőrizzük pl. az $x_2 = (4,3)$ mintára:

$$y_2{}^* = sign(-0{,}42 + 0{,}65 - 0{,}92) = -1 = y_2,$$

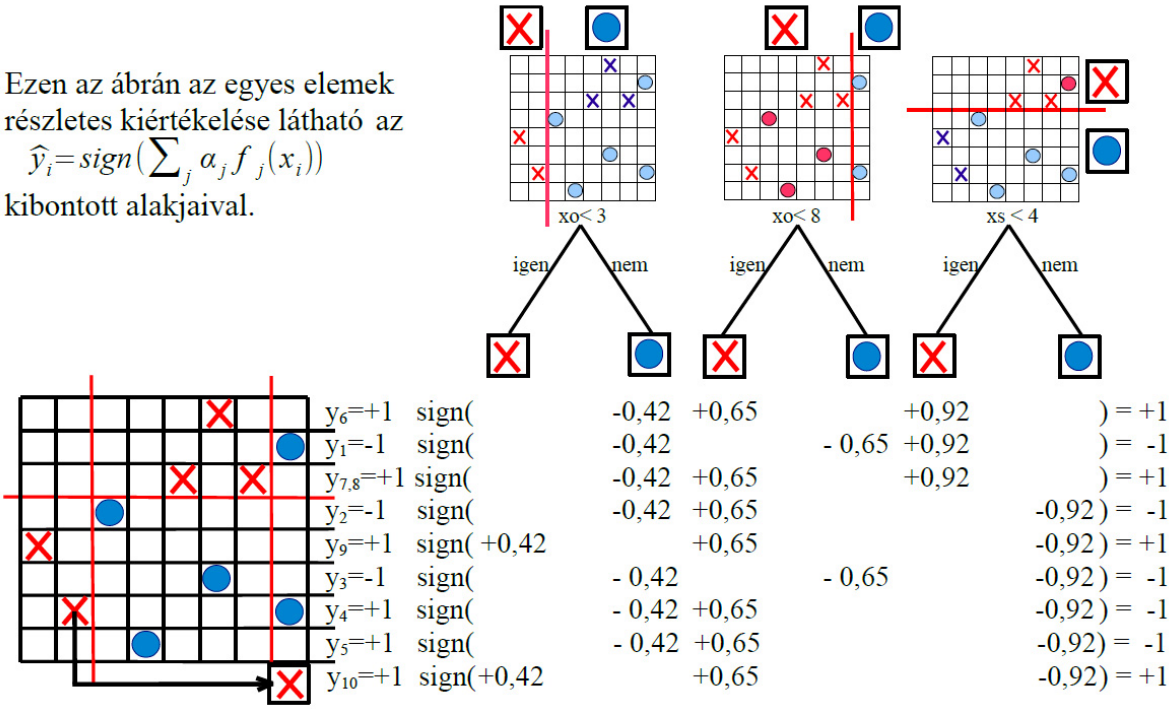
tehát helyes az eredmény. A tanulás mind a tíz mintára helyes eredményt ad, tehát a tanulás sikeres volt. Ez természetesen nem bizonyít semmit az egyéb pozíciók egyikére sem.



$\alpha_1 = 0{,}42$ $\alpha_2 = 0{,}65$ $\alpha_3 = 0{,}92$

A.6. ábra. Az eljárásnál alkalmazott döntési fák és azok súlyozása

Ezen az ábrán az egyes elemek részletes kiértékelése látható az $\hat{y}_i = sign\left(\sum_j \alpha_j f_j(x_i)\right)$ kibontott alakjaival.



A.7. ábra. A végeredmény részletes ellenőrzése a 10 mintaelemre

A.4. 10 kicsi indián feladat a GBM szemléltetéséhez

Amennyire könnyű jól végigkövethető példákat találni az AdaBoost eljáráshoz, annyira nehéz a szakirodalomban ugyanez a Gradient Boosting Modell (a továbbiakban GBM) esetén. A legtöbb bemutató feladat a GBM-t túlzottan leegyszerűsítve, annak csupán egy-egy mozzanatát mutatja be. A feladathoz motiváló példa található [8]. Ennek felhasználásával, továbbfejlesztésével egy másik adathalmazon mutatjuk be a GBM algoritmus lényegét.

Álljon a mintahalmazunk ismét 10 elemből: 10 kicsi indián, akikről elég keveset tudunk. A következő jellemzőket ismerjük:

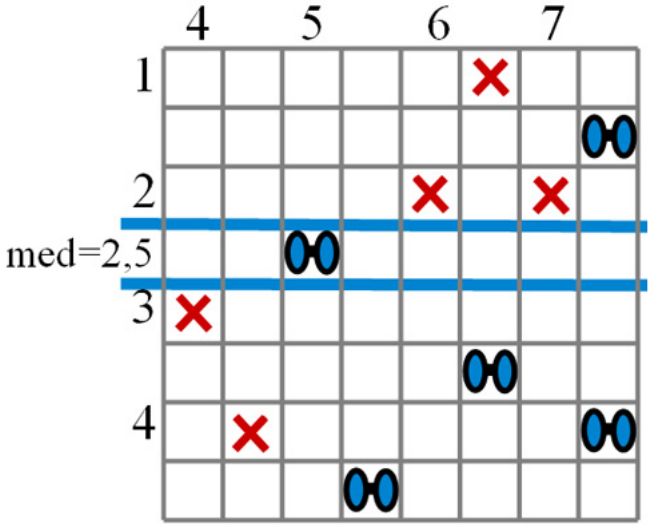
- id: azonosító
- ZH: a zárthelyi eredménye adatbányászatból
- oo: visel-e szemüveget (az oo inkább piktogram, mint rövidítés)
- láb: a lábuk (cipőjük) mérete

Feladatunk a ZH eredmény megbecsülése a többi jellemző alapján. *Az A.8.* ábrán az adatok kétféle ábrázolásban is láthatók. A baloldali ábrán az adatok a szokásos módon vannak elrendezve. Minden egyes elemhez hozzá vannak rendelve a magyarázóváltozók, valamint a célváltozó. A feladat a célváltozó értékének becslése. Ha a jobboldali grafikusabb ábrát nézzük, akkor meglepetéssel láthatjuk, hogy ez lényegében megegyezik a korábbi feladatunkkal. Azzal, hogy mást neveztünk ki célváltozónak, klasszifikációs feladatból regressziós feladatot kaptunk. A koordináták átírása valamilyen valóságosnak tűnő jellemzőre nem befolyásolja a feladatot, sem megoldási menetét.

Az adatokból ránézésre a következők látszanak:

- a két legnagyobb cipőméretű indián szemüveges
- a két legkisebb cipőméretű indián nem szemüveges
- Az előző két megállapítás lényegtelen, ezeket az adatokat ismerjük (nem a célváltozóra vonatkozó összefüggések, az előző klasszifikációs feladat első két lépése épp ezen tulajdonságokat használta fel)

id	ZH	oo	láb
70	1,5	1	7,5
71	2,5	1	5
72	3,5	1	6,5
73	4	1	7,5
74	4,5	1	5,5
75	1	0	6,5
76	2	0	6
77	2	0	7
78	3	0	4
79	4	0	4,5



A.8. ábra. A 10 kicsi indián feladat a táblás AdaBoost feladat megfordítása

- a 3-as fölötti ZH átlagú indiánok többsége szemüveges
- a 2,5 alatti átlagú indiánok többsége nem szemüveges

Vizsgáljuk meg a lehetséges döntési függvényeinket. Azt tapasztaljuk, hogy a függvények száma jócskán lecsökkent. Van hét döntési lehetőség a cipőméret alapján és egy a szemüveg alapján. Ez összesen nyolc három csúcsú (bináris) döntési fa. Használhatunk nagyobb mélységű fát, de a kérdések akkor is csak ezek lehetnek.

Egy GBM feladat kezdetekor dönteni kell a használni kívánt veszteségfüggvényről is. Az AdaBoost és változatainál ilyen lehetőség nem volt, mindegyik eljárás egy meghatározott veszteségfüggvényt optimalizált. Ugyanakkor azok a veszteségfüggvények csak a kategorikus változókhoz voltak alkalmazhatók. A veszteségfüggvényeket *a 2.5.* fejezetben ismertettük.

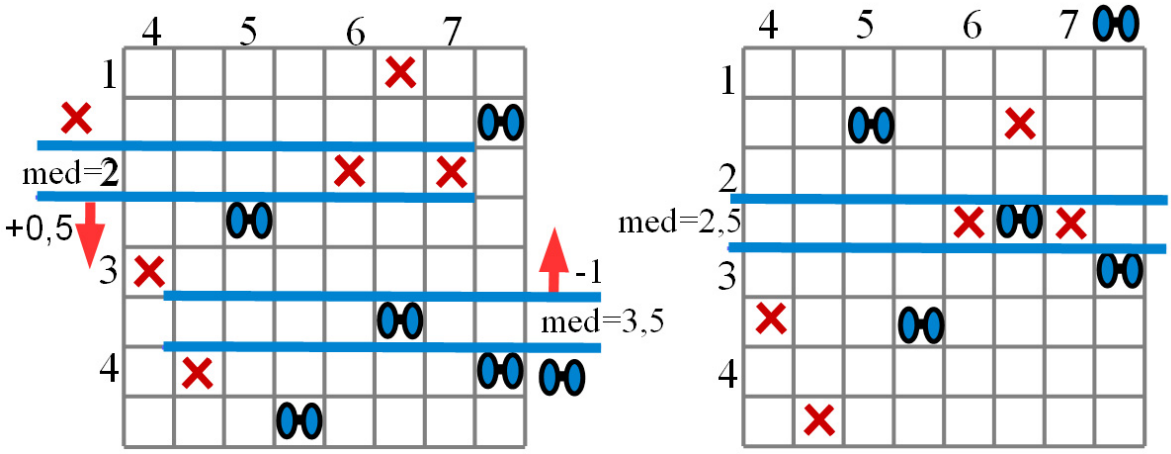
A.5. A 10 kicsi indián feladat grafikus megoldása

A következőkben kihasználva azokat az előnyöket, amelyeket a grafikus ábrázolás és a szigorú kvantálás ad, olyan grafikus szemléltetést, illetve megoldásmódot mutatunk be, amelynél semmilyen konkrét numerikus számításra nem lesz szükség, ha feltételként a hiba abszolút értékének a minimumát állítjuk be. Ehhez válasszuk az L_1 veszteségfüggvényt, aminek a minimuma a középső elemnél, vagyis a mediánnál van. A kis elemszám miatt

ez semmilyen nehézséget nem jelent. Vagyis az probléma, amit a medián meghatározása jelent nagy adathalmazon, az ilyen kis mennyiségnél nem mutatkozik. Példánkban a két középső elem 2,5 és 3, és így a függvény minimuma a [2,5 ; 3] tartományban van.

Nulladik lépésben becsüljük a minta elemeit egy konstans értékkel, ez legyen med = 2,5. Választhatnánk a med = 3 értéket is, a végeredmény ugyanaz lenne. Egyedül az lényeges, hogy a szabályos mediánképzés med = 2,75 középértékét ne válasszuk, mert azzal kilépnénk a kvantált értékeink rendszeréből. Ez a lépés már *az A.8.* ábrán fel lett tüntetve a medián szintvonalának ábrázolásával.

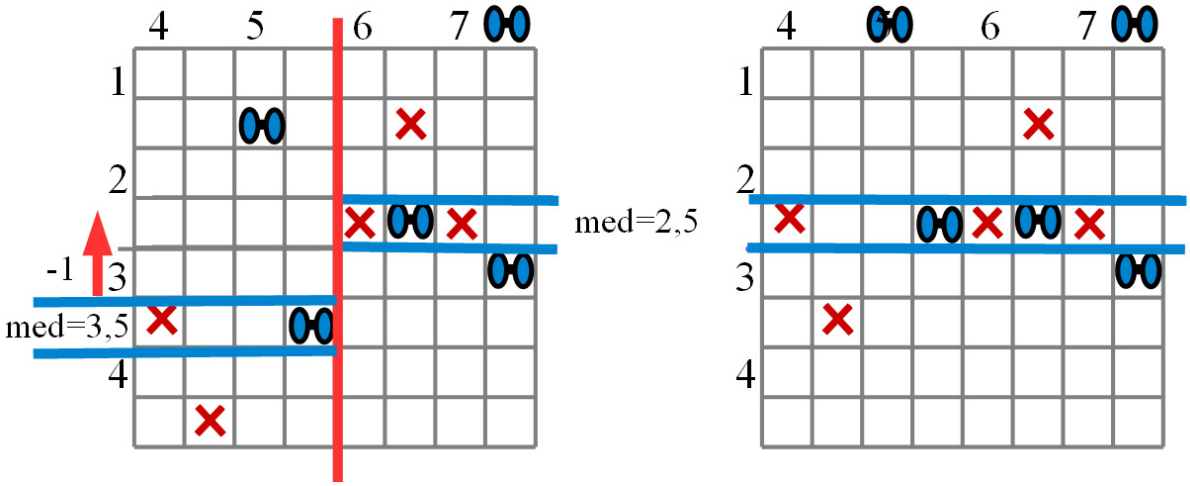
A további lépéseknél a minta kiválasztott elemeit fogjuk fel-le csúsztatni a táblán úgy, hogy ezen kiválasztott rész elemeinek mediánja rákerüljön az általunk 0-dik lépésben meghatározott 2,5-ös medián szintre.



A.9. ábra. Az eljárás 1. ciklusa. Az X és oo elemek mediánjai rákerülnek a szintvonalra

Első lépésben a szemüveg alapján választjuk két részre az elemeket. A medián képzése ilyen kis elemszám mellett nagyon egyszerű, a kiválasztott elemeket sorok szerint követve a középső elemet keressük. A nem szemüvegeseknél *(piros X)* ez a harmadik sor, így med = 2, ezen elemek célértékeit +0,5-tel megemelve a nem szemüvegesek mediánja is med = 2,5 lesz. Ez a baloldali ábra bal oldalán látható. Az öt szemüveges közül a harmadik a hatodik sorban található, vagyis ezek mediánja med = 3,5 (ua. ábra jobb oldalán). Itt az elemeket -1-gyel csökkentjük, ezek az elemek két sorral feljebb csúsznak. Az egyik elem ki is csúszik a tábláról, de ez nem lényeges, itt az elem célértéke 0,5. Az áttranszformált táblázat az ábra jobb oldalán látható.

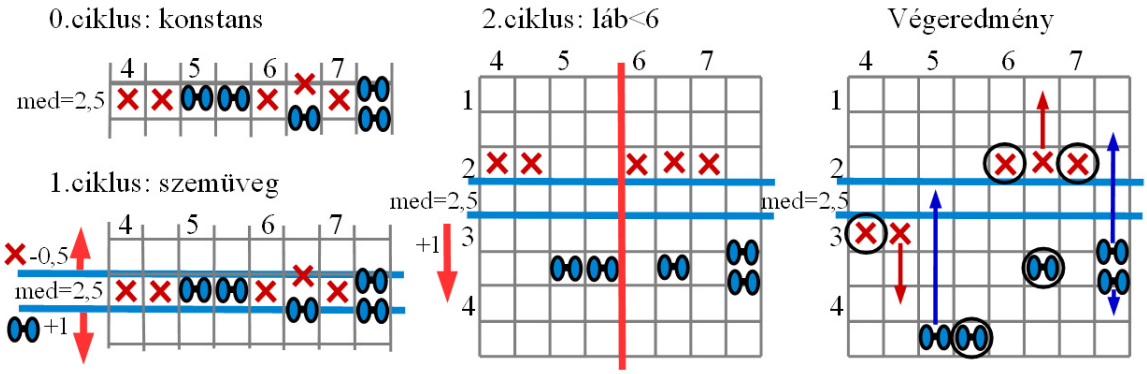
Második lépésben a lábméret alapján vágunk. Az optimális feltétel a láb > 6 kérdés. Ekkor külön vizsgáljuk a piros vonal egy-egy oldalát. A bal oldalon négy elemből a két középső a hatodik sorban található. Ekkor med = 3,5, vagyis ezen elemek értékeit 1-gyel csökkentjük, vagyis ezt az oldalt két sorral feljebb toljuk. Egy elem itt is kicsúszik a tábláról. A jobb oldal hat eleméből három (köztük a középső kettő) már a 2,5-ös medián szintnél áll. Ezzel az oldallal nincs további teendőnk.



A.10. ábra. Az eljárás 2. ciklusa. A baloldali elemek mediánja is felkerül a szintvonalra

Ezzel a végére is jutottunk a tanulási folyamatnak, vagyis az ábra jobb oldalán látható elrendezés már a végeredmény. Olyan eljáráshoz jutottunk, amely minden elemet pont akkora hibával becsül, amilyen távolságra van most a mediántól. Ez azt jelenti, hogy például a tábláról éppen kicsúszott elemek 4 lépésre, azaz ZH értékben 2 jegyre lesznek a tényleges értéküktől. Összességében az eljárás kezdetén a medián, mint abszolút hiba minimumának becslése összesen 10 ZH jegy értékű hibát adott. Az eljárás végén a hiba 6,5. Az éppen a medián vonalán álló 5 elem becslése pontos.

A kapott eljárás pontos meghatározásánál az eddigi lépések és eltolások lesznek a modell sarokkövei amik *az A.11.* ábra alapján is követhetők. Vegyük most a mintánk elemeit ZH eredményeik elfelejtésével, mindegyiket becsüljük a med = 2,5 értékkel. Ez a 0-dik lépés az ábra bal felső részén látható. A továbbiakban azokat a lépéseket ismételjük meg, amelyeket a tanulás során tettünk, csak épp a kiválasztott elemeket ellenkező irányba toljuk el. Ennek megfelelően az 1. ciklusnál jelezzük az X elemek +0,5-ös eltolását felfelé és az oo elemek -1-es eltolását lefelé. A 2. ciklusban a baloldali elemek (láb>6) +1-es eltolása zajlik le. Ezzel készen is vagyunk. Az utolsó jobboldali ábrán bekarikázva látszanak a pontosan becsült elemek, a többinél pedig egy nyíl mutatja, hová kellett volna érkezniük.



A.11. ábra. A ZH eredmények becslése a tanult eljárással

A.6. 10 kicsi indián a GBM algoritmussal

Gradient Boosting algoritmus

A.6.1. Megjegyzés. *A könnyebb áttekinthetőség kedvéért a 2.4. fejezetben ismertetett algoritmus szerepel itt is.*

Egy általános GBM algoritmus a következő lépéseket tartalmazza:

---

1. Bemenet:

(a) Legyen egy adott adathalmaz: $\{(x_i, y_i)\}_{i=1}^{N}$

(b) Választhatóak: M: interációs lépések száma

L: veszteségfüggvény

$h(x, \theta)$: alaptanuló függvények

(c) Kezdeti érték egy konstans:

$$f_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, \gamma)$$

2. Folyamat:

for $m = 1$ to M :

(a) for $i = 1, 2, \ldots, N$ képezzük:

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$$

(b) Illesszünk egy regressziós fát az r_{im} célváltozó-reziduálisokhoz, külön minden $R_{jm},\ j = 1, 2, \ldots, J_m$ levélhez.

(c) for $j = 1, 2, \ldots, J_m$

$$\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$

(d) Számoljuk újra: $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

3. Kimenet:

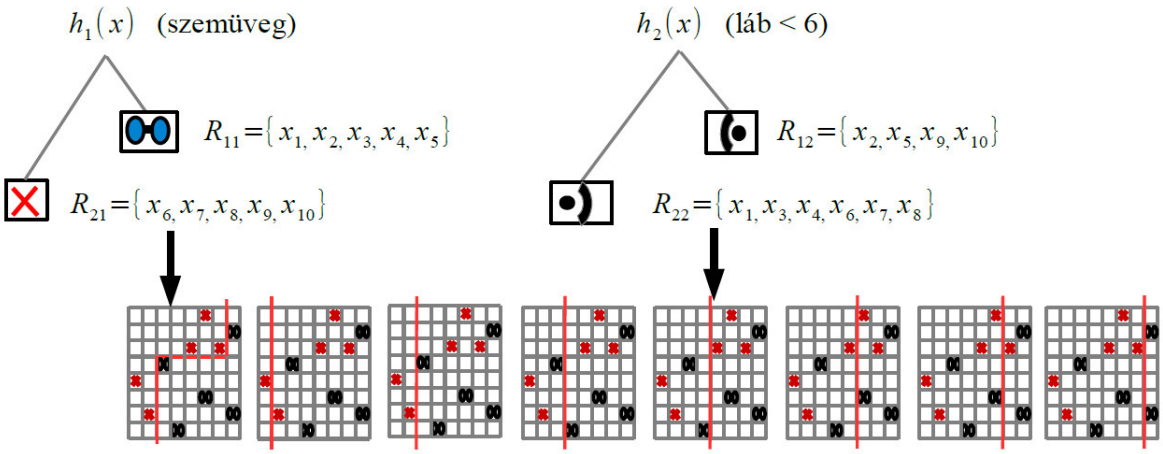
$f(x) = f_M(x)$

A következőkben vizsgáljuk meg, hogyan felel meg – ha megfelel – az előző végtelenül egyszerűnek tűnő tologatós játékunk egy általános GBM algoritmusnak. Szeretnénk bemutatni, hogy az előbbi egyszerű tologatós játék miért volt jó algoritmus az adott körülmények és a meghatározott optimalizálási célok között. Itt is – az AdaBoost algoritmus bemutatásához hasonlóan – az egyes lépéseknél ezen algoritmus számozására hivatkozunk.

Az egyes lépések az előző általános GBM algoritmus szám és betűjelzései szerint lesznek bemutatva. Az algoritmus számszerű eredményei lépésenként *az A.13.* ábra alapján követhetőek.

1. Bemenet - Bevezető lépések

(a) Legyen egy adott adathalmazunk célváltozóval, magyarázóváltozókkal. Ez a feltétel adott. *Az A.13.* ábra táblázatának legfelső sorában a mintahalmaz elemei vannak felsorolva. Az alkalmazott $h_1(x)$ és $h_2(x)$ függvények a mintahalmazt négy diszjunkt részhalmazra osztják. Ezen részhalmazok elemei - melyek az algoritmusban ugyanazt a becsült értéket kapják - egymás mellett kerülnek felsorolásra, eltérve a megszokott index szerint növekvő felsorolási konvenciótól. Az alatta lévő két sor alapján (az illeszkedési függvények és a magyarázó változók bemutatása) ez a felosztás jól követhető.



$risk(R_{1i})$	4	0	1	1,5	3	4	6,5	7,5
$risk(R_{2i})$	4,5	9,5	8	7,5	5	5	2,5	2,5
$\sum risk\ R$	8,5	9,5	9	9	8	9	9	10
$\Delta risk(R)$	-1,5	-0,5	-1	-1	-2	-1	-1	0

A.12. ábra. Az eljárásban használt illeszkedési függvények kiválasztása

(b) Határozzuk meg a ciklusszámot, az illeszkedési függvényeinket, továbbá az alkalmas veszteségfüggvényt. Ezek tulajdonképpen már az előzőekben meghatározásra kerültek, tehát a ciklusszám $T = 2$, L_1 veszteségfüggvényt használunk és az illeszkedési függvényeket már ott számbavettük. *Az A.12.* ábra felső részén a két alkalmazandó függvény látható.

Ugyanakkor lentebb az összes lehetséges vágás mindkét oldalának és azok összegének kockázatértékei vannak feltüntetve. Ez alapján látható, hogy ez a két vágás adja a legkisebb kockázatértéket.

(c) Kezdeti érték: $f_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} L_1(y_i, \gamma)$

L_1 veszteségfüggvény a mediánnál veszi fel a minimum értékét, ha két középső elem van, akkor e két elem között végig a minimumot adja. Itt med = 2,5 lesz a kezdőérték. A $\sum_{i=1}^{10} L_1(y_i, \gamma)$ függvény optimalizálása *az A.13.* ábra alján baloldalt látható.

2. Folyamat - Főciklus

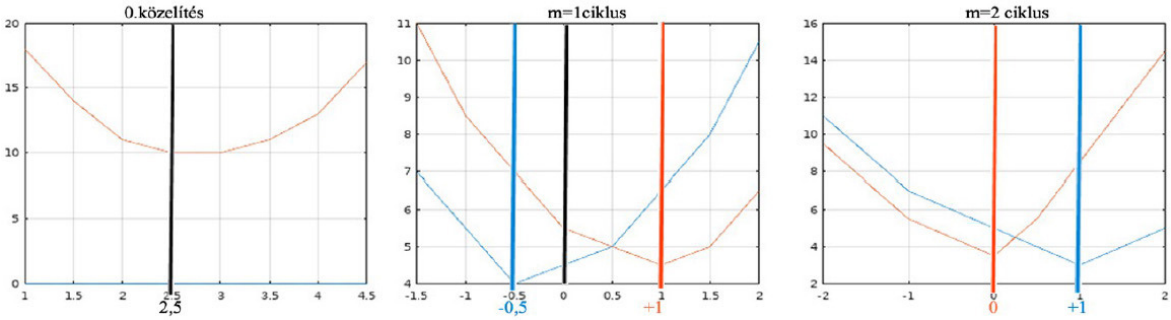
A két főciklus lépései is jól követhetőek az ábráról. A h_1 és h_2 háromcsúcsú fafüggvények leveleinek optimalizálása az ábra alján található. Felmerül a kérdés, hol alkalmazzuk itt a gradiens módszert. A gradiens módszert a

$$\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$

kifejezések megoldásához használnánk, ami L_1 veszteségnél a mediánt adja, amit itt egyszerű ránézéssel is meg tudunk határozni.

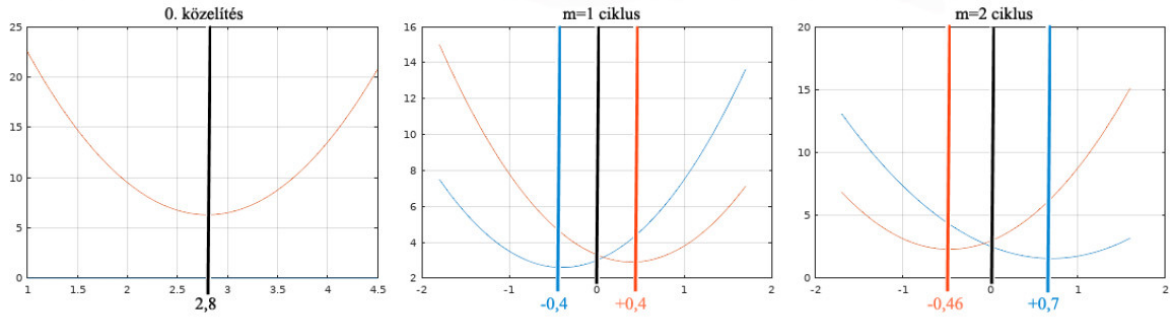
A grafikusan bemutatott tologatós algoritmus eltolásai megfelelnek az itt meghatározott $\gamma_j m$ értékeknek, amelyekkel az $f_m(x)$-t korrigáljuk és így a reziduális tagokat is hasonlóképpen. Ugyanakkor a tologatós algoritmusnál, ahol a medián sorától való távolság egyenlő a becslés abszolút hibájával, ezt nem jelenítettük meg a függőleges skálán, vagyis nem vontuk ki a med = 2,5 becsült értéket a reziduálisokból. Igaz viszont, hogy egyáltalán nincs szerepe, milyen értékek szerepelnek ezen a skálán, az a kijelölt mediánvonal és a részhalmazok mediánjának viszonyát nem befolyásolja. Felmerül a kérdés, mi történik, ha L_1 helyett L_2 veszteségfüggvényt alkalmazunk, vagyis a becslés négyzetes hibáját minimalizáljuk. Ekkor hasonló, de az előzőtől eltérő eredményt kapunk, ami *az A.14.* ábrán követhető.

(1) Bevezető lépések											
minták		x_2	x_5	x_9	x_{10}	x_6	x_7	x_8	x_1	x_3	x_4
Mintatér particionálása $h_1(x)$ és $h_2(x)$ háromcsúcsú döntési függvényekkel		OO		X					OO		
		láb<6				*láb*≥6					
y_i célértékek		2,5	4,5	3	4	1	2	2	1,5	3,5	4
L_1 veszteségfüggvény és a 0-dik közelítés : $f_0(x)=argmin_\gamma \sum_{i=1}^{10} L_1(y_i,\gamma)=med(Y)=2,5$											
$\sum_{i=1}^{10} L_1(y_i,f_0)=10 \qquad (y_i-f_0)$		0	2	0,5	1,5	-1,5	-0,5	-0,5	-1	1	1,5
(2) Főciklus m=1 ciklusa											
(a)képezd: $r_{i1}=sign(y_i-f_0)$		0	1	1	1	-1	-1	-1	-1	1	1
(b) Illessz egy regressziós fát az r_{i1} célváltozó-reziduálisokhoz, külön minden R_{j1}, j = 1, 2 levélhez.											
(c) for j = 1, 2: $\gamma_{j1}=med(f_0(x_i))_{x_i\in R_{j1}}$	R_{11}	1							1		
	R_{21}			-0,5							
(d) Számold újra: $f_1(x)=f_0(x)+\sum_{j=1}^{2}\gamma_{j1}I(x\in R_{j1})$		3,5	3,5	2	2	2	2	2	3,5	3,5	3,5
$\sum_{i=1}^{10} L_1(y_i,f_1)=8,5 \qquad (y_i-f_0)$:		-1	1	1	2	-1	0	0	-2	0	0,5
(2) Főciklus m=2 ciklusa											
(a)képezd: $r_{i2}=sign(y_i-f_1)$		-1	1	1	1	-1	0	0	-1	0	1
(b) Illessz egy regressziós fát az r_{i2} célváltozó-reziduálisokhoz, külön minden R_{j2}, j = 1, 2 levélhez.											
(c) for j = 1, 2: $\gamma_{j2}=med(f_1(x_i))_{x_i\in R_{j2}}$	R_{12}	1									
	R_{22}						0				
(d) Számold újra: $f_2(x)=f_1(x)+\sum_{j=1}^{2}\gamma_{j2}I(x\in R_{j2})$		4,5	4,5	3	3	2	2	2	3,5	3,5	3,5
$\sum_{i=1}^{10} L_1(y_i,f_2)=6,5 \qquad (y_i-f_2)$:		-2	0	0	1	-1	0	0	-2	0	0,5



A.13. ábra. Indian10 GBM számítások teljes menete L_1 veszteségfüggvénnyel

(1) Bevezető lépések											
minták		x_2	x_5	x_9	x_{10}	x_6	x_7	x_8	x_1	x_3	x_4
Mintatér particionálása $h_1(x)$ és $h_2(x)$ háromcsúcsú döntési függvényekkel		OO		X					OO		
		láb<6				*láb*≥6					
y_i célértékek		2,5	4,5	3	4	1	2	2	1,5	3,5	4
L_2 veszteségfüggvény és a 0-dik közelítés : $f_0(x)=argmin_\gamma \sum_{i=1}^{10} L_2(y_i,\gamma)=E(Y)=2{,}8$											
$\sum_{i=1}^{10} L(y_i,f_0)=6{,}3$ (y_i-f_0) :		-0,3	1,7	0,2	1,2	-1,8	-0,8	-0,8	-1,3	0,7	1,2
(2) Főciklus m=1 ciklusa											
(a)képezd: $r_{i1}=(y_i-f_0)$		-0,3	1,7	0,2	1,2	-1,8	-0,8	-0,8	-1,3	0,7	1,2
(b) Illessz egy regressziós fát az r_{i1} célváltozó-reziduálisokhoz, külön minden R_{j1}, j = 1, 2 levélhez.											
(c) for j = 1, 2: $\gamma_{j1}=E(f_0(x_i))_{x_i\in R_{j1}}$	R_{11}	0,4							0,4		
	R_{21}			-0,4							
(d) Számold újra: $f_1(x)=f_0(x)+\sum_{j=1}^{2}\gamma_{j1}I(x\in R_{j1})$		3,2	3,2	2,4	2,4	2,4	2,4	2,4	3,2	3,2	3,2
$\sum_{i=1}^{10} L_2(y_i,f_1)=5{,}5$ (y_i-f_1) :		-0,7	1,3	0,6	1,6	-1,4	-0,4	-0,4	-1,7	0,3	0,8
(2) Főciklus m=2 ciklusa											
(a)képezd: $r_{i2}=(y_i-f_1)$		-0,7	1,3	0,6	1,6	-1,4	-0,4	-0,4	-1,7	0,3	0,8
(b) Illessz egy regressziós fát az r_{i2} célváltozó-reziduálisokhoz, külön minden R_{j2}, j = 1, 2 levélhez.											
(c) for j = 1, 2: $\gamma_{j2}=E(f_1(x_i))_{x_i\in R_{j2}}$	R_{12}	0,7									
	R_{22}						-0,47				
(d) Számold újra: $f_2(x)=f_1(x)+\sum_{j=1}^{2}\gamma_{j2}I(x\in R_{j2})$		3,9	3,9	3,1	3,1	1,93	1,93	1,93	2,73	2,73	2,73
(y_i-f_2)		-1,4	0,6	-0,1	0,9	-0,93	0,07	0,07	-1,23	0,77	1,27



A.14. ábra. Indian10 GBM számítások teljes menete L_2 veszteségfüggvénnyel

A.7. Az AdaBoost, MadaBoost modell programjának forráskódja

```
%adaboost.m
%adatok:
adakod=2;       % 0:ada  1: mada  2:filter
tt=3;                   % ciklusok száma
%************************************
%bemenő adatok
%************************************
X1=[ 2  4  6  7  8  1  3  3  5  7 ]; % a minták sorpoziciói
X2=[ 8  3  6  8  4  6  5  7  1  2 ]; %a minták oszloppoziciói
Y= [-1 -1 -1 -1 -1  1  1  1  1  1 ]; %a minták besorolása:O:-1 X:+1
A = [X1;X2;Y];
fter=alaptanulo(1,A);
[fszam,f2]=size(fter);
suly(1:10)= 0.1;
suly0=suly;
sulynaplo(1,4:13)=suly;
t=1;
%*********************************************
prod_hixfx(1:tt,1:10)=zeros;
prod_hixfx(1,1:10)=ones;
for t=1:1:tt
    suly_ert=suly.*A(3,1:10);
    var_ert(1:10)=zeros;
    for i=1:1:10
       var_ert=var_ert+suly_ert(i);
    end
    err_min=1;
for i=1:1:fszam
    f_ert=fter(i,3:12).*suly_ert;
    sum=0;
    for j=1:1:10
        sum=sum+f_ert(j);
    end
    err_k=(sum+1)/2;
```

```
    aa(t,i)=err_k;
    aa(t,i+fszam)=1-err_k;

    if err_min>err_k
        err_min=err_k;
        minhely=i;
        piros=-1;
    end
    if err_min>1-err_k
        err_min=1-err_k;
        minhely=i;
       piros=1;
    end
end
%--------------------------------
    alfa(t)=0.5*log((1-err_min)/err_min);
    wplusz= exp(alfa(t));
    wminusz= exp(-alfa(t));
    elojel=piros*fter(minhely,3:12).*T(3,1:10);
    hix(t,1:10)=piros*fter(minhely,3:12);
    for i=1:1:10
        if elojel(i)==1
            prod0(i)= wminusz;
        else
            prod0(i)=wplusz;
        end
    end
    prodnext=prod0.*prod_hixfx(t,1:10);
%--------------------------
if adakod==1 % MadaBoost
    for i=1:1:10
        if( prodnext(i)>1)
            prodnext(i)=1;
        end
    end
end
%--------------------------
if adakod==2 % FilterBoost
```

```
    sum_hix(1:10)=zeros;
    for i=1:1:t
        sum_hix=sum_hix+(alfa(i)*hix(i,1:10));
    end
    wm=1/(1+exp(1));
    wp=wm*exp(1);
     for i=1:1:10
         if sum_hix(i)*T(3,i)>0
             prodnext(i)=wm;
         else prodnext(i)=wp;
         end
     end
     prodnext=prodnext.*suly;
end
%---------------------------
suly=prodnext.*suly0;
oszsuly=0;
for i=1:1:10
    oszsuly= oszsuly+suly(i);
end
suly=(1/oszsuly).*suly;
prod_hixfx(t+1,1:10)=prodnext;

sulynaplo(2*t,1:3)=[err_min,minhely,piros];
sulynaplo(2*t,4:13)=prodnext;
sulynaplo(2*t+1,1:3)=[alfa(t),wplusz,wminusz];
sulynaplo(2*t+1,4:13)=suly(1:10);
end
```

A meghívott függvények forráskódja:

```
function [ fter] = alaptanulo( ff,minta )
%fuggvenyter reprezentáció
%fter:
%kimenet:fter matrix sorai a match fuggveny parameterei és értékei
%ff=1;    % 1 vágás: 2x2  2:3x3  3: 5x5
%************************************
%bemenő adatok
```

```
%************************************
%X1=[ 2  4  6  7  8  1  3  3  5  7 ];
%X2=[ 8  3  6  8  4  6  5  7  1  2 ];
%Y= [-1 -1 -1 -1 -1  1  1  1  1  1 ];
%minta = [X1;X2;Y];
%*********************************************
[M,N]=size(minta);

%vertikális és horizontális vágások
if ff==1

    for i=1:1:2
        for j=1:1:7
          sum=0;
          match(1:N)=zeros;
          for k=1:1:N
                if i==2
                    if (minta(1,k)<j+1)
                    match(k)=1;
                    else
                    match(k)=-1;
                    end
                elseif (minta(2,k)<j+1)
                    match(k)=1;
                else
                    match(k)=-1;
                end
            end
            fter(7*(i-1)+j,1:N+2)=[j,i,match];
        end
    end
end

% 3x3-as négyzetek
if ff==2
    for i=1:1:8
        for j=1:1:8

```

```
            match(1:N)=zeros;
            for k=1:1:N
                if ((abs(minta(1,k)-i)<2)&&(abs(minta(2,k)-j)<2))
                    match(k)=1;
                else
                    match(k)=-1;
                end
            end
            fter(8*(i-1)+j,1:N+2)=[i,j,match];

        end
    end
end
% 5x5-ös négyzetek
if ff==3
    for i=1:1:8
        for j=1:1:8

            match(1:N)=zeros;
            for k=1:1:N
                 if ((abs(minta(1,k)-i)<3)&&(abs(minta(2,k)-j)<3))
                  match(k)=1;
                else
                  match(k)=-1;
                end
            end
            fter(8*(i-1)+j,1:N+2)=[i,j,match];

        end
    end
end

end
```

Argminloss függvény:

```
function [yk0,yk1] = argminloss ( TT,loss )
v=size(TT);
```

```
mintaszam=v(2);
%várható érték Ex
Ex=0;
Ex2=0;
for i=1:1:mintaszam
    Ex=Ex+TT(2,i);
    Ex2=Ex2+(TT(2,i))^2;
end
Ex=Ex/mintaszam;
Ex2=Ex2/mintaszam;
D2x= Ex2-Ex^2;
Dx=sqrt(D2x);
%-------------
xv=TT(1,1:mintaszam);
yv=TT(2,1:mintaszam);
y_min=min(yv);
y_max=max(yv);
lep=(y_max-y_min)/1000;
err0_min=100000;
err1_min=100000;

for i=1:1:1001
    ya=y_min+(i-1)*lep;
    err1=0;
    err0=0;
    for j=1:1:mintaszam
        if loss==1
            losx=abs(ya-yv(j))
        else
            losx=0.5*(ya-yv(j))^2;
        end
        % ha Huber loss
        if (loss==3)&&(losx>0.5*D2x)
            losx=Dx*abs(ya-yv(j))-0.5*D2x;
        end
        % ha Tukey loss
        if (loss==4)&&(losx>0.02*D2x)
          losx=0.02*D2x;
```

```
        end
        if xv(j)==1
          err1=err1+losx;
        else
          err0=err0+losx;
        end
    end
    if err1<err1_min
        err1_min=err1;
        yk1=ya;
    end
    if err0<err0_min
        err0_min=err0;
        yk0=ya;
    end
    err1func(i)=err1;
    err0func(i)=err0;
end
for i=1:1:mintaszam
    if xv(i)==1
        yv(i)=yv(i)-yk1;
    else
        yv(i)=yv(i)-yk0;
    end
end
endfunction
```

B. függelék

Jelölések, jegyzékek

B.1. Jelölések

x	változó
$\mathbf{x}$	vektor
$\mathbf{A}$	mátrix
$\mathbf{I}$	identitás mátrix
$\mathcal{X}, \mathcal{Y}$	bemeneti, kimeneti tér (magtér, képtér)
$\mathcal{D}$	valószínűségi eloszlás
D	adatminta (adathalmaz)
$\mathcal{N}$	normális eloszlás
$\mathcal{U}$	egyenletes eloszlás
$\mathcal{H}$	hipotézis tér
H	hipotézisek halmaza
$h(\cdot)$	hipotézis (tanuló)
$\mathcal{L}$	tanuló algoritmus
$p(\cdot)$	valószínűségi sűrűség függvény
$p(\cdot \mid \cdot)$	feltételes valószínűségi sűrűségfüggvény
$P(\cdot)$	valószínűségi eloszlás függvény
$P(\cdot \mid \cdot)$	feltételes valószínűségi eloszlás függvény
$\mathbb{E}_{\cdot \sim D}[f(\cdot)]$	a $\mathcal{D}$ eloszlásbeli $f(\cdot)$ függvény várható értéke
$var_{\cdot \sim D}[f(\cdot)]$	a $\mathcal{D}$ eloszlásbeli $f(\cdot)$ függvény szórása
$\mathbb{I}(\cdot)$	indikátor függvény: 1 ha $\cdot$ igaz; 0, ha $\cdot$ nem igaz
$\text{sign}(\cdot)$	előjel függvény: -1, 1 vagy 0 ha $\cdot < 0$, $\cdot > 0$, vagy $\cdot = 0$
$err(\cdot)$	hiba függvény

$\{\ \ldots\ \}$	halmaz
$(\ \ldots\ )$	sor vektor

B.2. Magyar –angol szójegyzék

magyar	**angol**
abszolút többségi szavazás	majority voting
additív logisztikus regressziós modellek	additive logistic regression modell
additív súlyozott kombináció	additive weighted combination
alapigazság-függvény	ground-truth function
diszkrét, kategorikus	nominal
döntési fa	decision tree
egyszerű átlag	simple averaging
egyszerű lágy szavazás	simple soft voting
első szintű tanuló	first-level learner
elutasítási mechanizmus	rejection option
exponenciális veszteségfüggvény	exponential loss function
éles címke	crisp label
értelmességi kritérium ellenőrzése	sanity check
fa alapú hibrid módszerek	bagging
független tanuló	individual learner, component learner
gyenge (alap-, primitív) tanuló, modell	weak learner, base learner, base learning algorithm
gyorsítás	boost
három csúcsú (bináris) döntési fa	tree stamps
helyettesítő függvény	surrogate function
helyettesítő veszteségfüggvény	surrogate loss function
heterogén modellsorozatok	heterogeneous ensembles
hibrid modellek	ensemble methods, commitee-based learning, multiple classifier systems
hibrid modellek nyesése	ensemble pruning
homogén modellsorozatok	homogeneous ensembles
izotónikus regresszió	isotonic regression
kalibrációs módszerek	calibration methods
klasszifikációs hiba	classification error
klasszifikátor	classifier
klasszifikátor alapú súlyozás	classifier specific weight
kockázatfüggvény	risk function
korai befejezési probléma	early-termination problem
lágy szavazás	soft voting
másodszintű tanuló	second-level learner, meta-larner

mintaelem (adatpont) alapú súlyozás	class-specific weight
nagy torzítás	high bias
nehézségi osztály	complessity classis
neurális háló	neural network
osztály alapú súlyozás	class specific weight
osztályvalószínűség	class probability
platt méretezés	platt scaling
posterior valószínűség	posterior probability
relatív többségi szavazás	plurality voting
stabil	robust
statisztikai probléma	statistical issue
súlyozott átlag	weighted averaging
súlyozott lágy szavazás	weighted soft voting
súlyozott szavazás	weighted voting
szavazás	voting
számítási idő probléma	computational issue
szórásnégyzet	variance
tartóvektor gép	support vector machine
újramintavételezés	re-sampling
újrasúlyozás	re-weighting
veszteségfüggvény	veszteség function
véletlen tippelés, $0R$ algoritmus	random guess, $0R$ algorithm

B.3. Ábrák jegyzéke

1.1. A hibrid modellek fajtái 8

1.2. A hibrid modellek felépítése 9

2.1. Az AdaBoost tanító- és teszt hibájának szemléltetése 15

2.2. súlyozások alakulása 16

2.3. MadaBoost eljárás együtthatói 17

2.4. Nielsen különböző eljárásokkal közelíti a konstruált mintafüggvényt 20

2.5. A Huber- és Tukey- veszteségfüggvény összehasonlítása 24

2.6. Kategorikus lossok 25

4.1. A kombinációs módszerek fajtái 31

A.1. A tiny toy és a táblás modell lényegében azonosak 42

A.2. Több gyenge osztályozás összessége kiváló eredményt is adhat 42

A.3. AdaBoost kezdeti lépések: adatok és függvények. Kezdeti súlyozás 45

A.4. Az AdaBoost eljárás első két ciklusa. Vágások és átsúlyozások 46

A.5. Az AdaBoost eljárás harmadik ciklusa és a végeredmény 47

A.6. Az eljárásnál alkalmazott döntési fák és azok súlyozása 48

A.7. A végeredmény részletes ellenőrzése a 10 mintaelemre 48

A.8. A 10 kicsi indián feladat a táblás AdaBoost feladat megfordítása 50

A.9. Az eljárás 1. ciklusa. Az X és oo elemek mediánjai rákerülnek a szintvonalra 51

A.10. Az eljárás 2. ciklusa. A baloldali elemek mediánja is felkerül a szintvonalra 52

A.11. A ZH eredmények becslése a tanult eljárással 53

A.12. Az eljárásban használt illeszkedési függvények kiválasztása 55

A.13. Indian10 GBM számítások teljes menete L_1 veszteségfüggvénnyel 57

A.14. Indian10 GBM számítások teljes menete L_2 veszteségfüggvénnyel 58

B.4. Algoritmusok jegyzéke

2.1. Általános Boosting algoritmus 12

2.2. AdaBoost algoritmus 14

2.3. Gradient Boosting algoritmus 19

2.4. LogitBoost algoritmus 27

2.5. LPBoost algoritmus 28

3.1. Bagging algoritmus 30

4.1. Átlag 33

4.2. Szavazás 35

4.3. Tanulók általi kombinálás általánosan 40

B.5. Irodalomjegyzék

[1] Chen, Tianqi ; Guestrin, Carlos: *XGBoost: A Scalable Tree Boosting System.* San Francisco, 2016.ls

[2] Chen, Tianqi: *Introduction to Boosted Trees.* University of Washington, Oct. 22.2014 `https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf` (2017.10.23)

[3] Dasarathy,B.V.; Sheela, B.V.: *A composite classifier system design: Concepts and methodology.* Proceedings of the IEEE (Volume: 67, Issue: 5, May 1979)

[4] Dong, Ying: *Beating Kaggle the easy way (MsC szakdolgozat).* Darmstadt, 2015. `https://www.ke.tu-darmstadt.de/lehre/arbeiten/studien/2015/Dong_Ying.pdf` (2017.10.23)

[5] Efimov, Dmitry: *Gradient Boosting Trees: theory and applications.* 2016. `http://efimov-ml.com/pdfs/Lvov_XGBoost_2016.pdf` (2017.10.23)

[6] Friedman, Jerome H.; Popescu, Bogdan E.: *Predictive Learning via Rule Ensembles.* Stanford, 2005. `http://statweb.stanford.edu/~jhf/ftp/RuleFit.pdf` (2017.10.23)

[7] Friedman, Jerome H.; Popescu, Bogdan E.: *Importance Sampled Learning Ensembles.* September 9, 2003. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.2.5171&rep=rep1&type=pdf` (2017.10.23)

[8] Gorman, Benjamin: Gormanalysis Jan 12, 2017 `https://gormanalysis.com/gradient-boosting-explained/` (2017.11.30)

[9] Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Second ed. Springer, Stanford, 2008.

[10] He, Tong: *XGBoost: eXtreme Gradient Boosting.* `http://www.saedsayad.com/docs/xgboost.pdf` (2017.10.23)

[11] Huber, Peter: *Robust Estimation of a Location Parameter* in: The Annals of Mathematical Statistics Vol.35 N 1 (1964) p.73-101. `https://projecteuclid.org/euclid.aoms/1177703732` (2017.12.17)

[12] James, Gareth; Witten, Daniela; Hastie, Trevor; Tibshirani, Robert: *An Introduction to Statistical Learning with Applications in R.* Springer, New York, 2013.

[13] Kearns, Michael J.;Vazirani, Umesh V.: *An Introduction to Computational Learning The- ory.* MIT Press, 1994.

[14] Korolev, Maksim; Ruegg, Kurt: *Gradient Boosted Trees to Predict Store Sales.* Stanford, Harward, 2015.

[15] Li, Cheng: *A Gentle Introduction to Gradient Boosting.* `http://www.ccs.neu.edu/home/vip/teach/MLcourse/4_boosting-/slides/gradient_boosting.pdf` (2017.10.23)

[16] Natekin, Alexey; Knoll, Alois: *Gradient Boosting Machines, a Tutorial.* frontiers in NEUROROBOTICS method article. München, Germany, 2013.12.04 `https://www.researchgate.net/publication/259653472_Gradient_Boosting_Machines_A_Tutorial` (2017.10.23)

[17] Nielsen, Didrik: *Tree Boosting with XG Boost.* NTNU Norwegian University of Science and Technology, 2016.

[18] Ramraj, Sunil; Nishant, Uzir; Banerjee, Shatadeep: *Experimenting XGBoost Algorithm for Prediction and Classification of Different Datasets.* in: International Journal of Control Theory and Applications Volume 9 Number 40 2016. `https://www.researchgate.net/publication/318132203_XGBoost_Algorithm_for_Prediction_and_Classification_of_Different_Datasets` (2017.10.23)

[19] Schapire, Robert E.; Freund, Yoav: *Boosting Foundations and Algorithms.* The MIT Press, Cambridge Massachusetts London England, 2012.

[20] Schapire, Robert E.: *The Strength of Weak Learnability Machine Learning.* 5, p.197-227 1990.

[21] Seni, Giovanni; Elder, John: *Ensemble Methods in Data Mining: Improving Accuracy Through Combining Predictions.* Morgan & Claypool Publishers, Chicago, 2010.

[22] Tukey, J.W.: *Exploratory Data Analysis.* Addison-Wesley, Reading, MA. 1977

[23] Tan, Pang-Ning; Steinbach, Michael; Kumar, Vipin: *Adatbányászat/Bevezetés az adatbányászatba.*Panem, Budapest 2011. `http://www.tankonyvtar.hu/hu/tartalom/tamop425/0046_adatbanyaszat/index.html` (2017.11.17)

[24] Valiant, Leslie: *A theory of the learnable.* Communications of the ACM Vol.27 No.11 p. 1134-1142. 1984 Nov.

[25] Valiant, Leslie: *Probably Approximately Correct – Nature's Algorithms for Learning and Prospering in a Complex World*. 2013.

[26] Viola, Paul; Jones, Michael: *Rapid Object Detection using a Boosted Cascade of Simple Features*.ACCEPTED CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2001.
`https://wearables.cc.gatech.edu/paper\_of\_week/viola01rapid.pdf` (2017.12.30)

[27] Zhang, Huan; Si, Si; Hsieh, Cho-Jui: *GPU-acceleration for Large-scale Tree Boosting*. California, 2017.

[28] Zhou, Hucheng; Zhao, Bo: *SparkTree: Push the Limit of Tree Ensemble Learning*. Beijing, China; Nanjing, China, 2016.
`https://www.microsoft.com/en-us/research/wp-content/uploads/2017/04/main.pdf` (2017.10.23)

[29] Zhou, Zhi-Hua: *Ensemble Methods: Foundations and Algorithms*. CRC Press Taylor & Francis Group, New York, 2012.